XML Messaging with JAXM

Reviewed: SavaJe XE

JAVAPro

A Fawcette Technical Publication January 2002 Vol. 6 No. 1 www.javapro.com

Open! Scalable! Firewall Friendly!

The Power of SOAP

Cross-Platform, Cross-Language Communication

Get Extreme With Automated Unit Testing

HTML File Upload Made Easy

Java and .Net Compared

Free Source Code

Code Online www.javapro.com

U.S. $5.95 (Canada $7.95)

01>

7 25274 91607 8

Net Wt. 94 oz (2.66 kg)

I guess you haven't heard of TogetherSoft!

*"Using Together ControlCenter, we were able to take an application from **beginning to end in only 48 days.** It was one of the quickest deliveries of an application to date."*

*— Kevin Monaghan, Consultant, **General Motors Corporation***

Award-winning technology from TogetherSoft:

Java Developer's Journal Finalist
Best Java Modeling Tool
Best Java Application
Best Java Reporting Tool
Together ControlCenter
TogetherSoft Corporation 2001

Software Development
11th Annual Productivity Award
Together ControlCenter
TogetherSoft Corporation 2001

Java Developer's Journal
Best Java Application
Together/J™
TogetherSoft Corporation 2000

Java Pro 2000 Readers' Choice
Most Innovative Java Product
Together/J
TogetherSoft Corporation 2000

NC Tech Fast 50
#4 Five-Year Growth Rate
TogetherSoft Corporation
2001

You don't have to work all night.

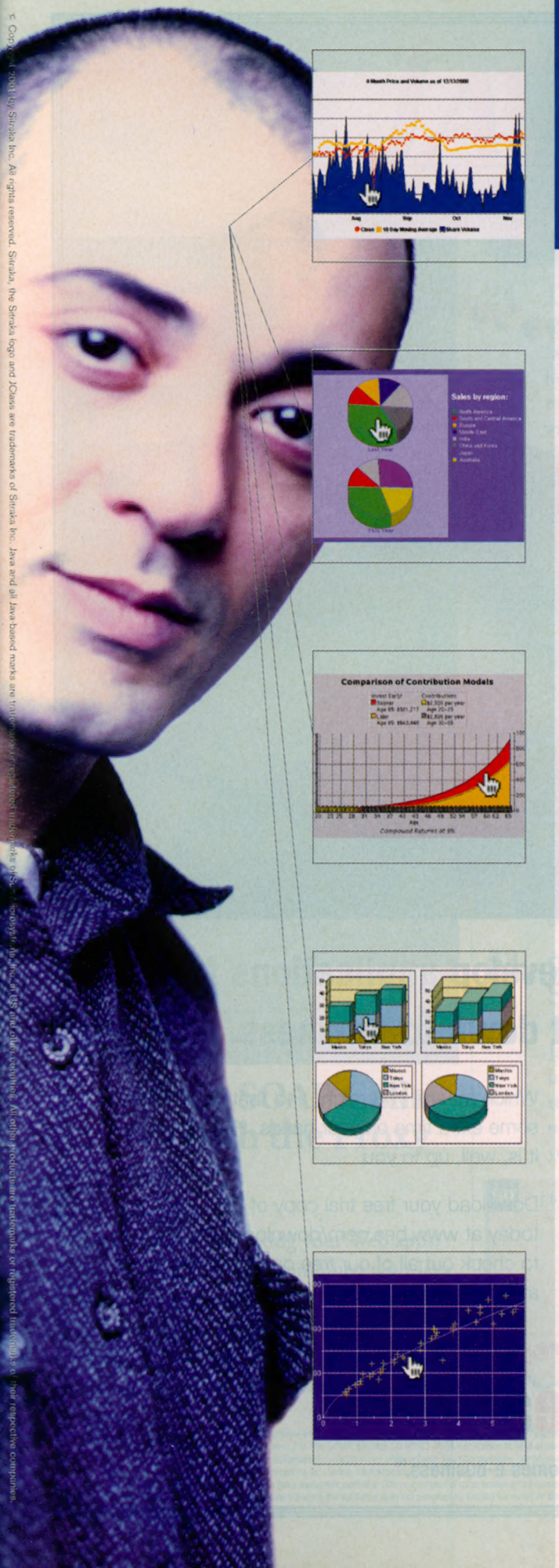
A staggering 84% of software projects come in late and over budget. Avoid being a part of that statistic. Cut your development time with Together® ControlCenter™.*

Download Together ControlCenter today and experience it for yourself. For a special extended eval period, 30 days instead of our usual 15, use this URL: www.togethersoft.com/1/jp.jsp

For more information **Tel (919) 833-5550**
900 Main Campus Drive, Suite 500, Raleigh, NC 27606 USA

TogetherSoft™
Improving the ways people work together™

©2001 TogetherSoft Corporation. All rights reserved. Patents pending. TogetherSoft, the TogetherSoft Logo, Together ControlCenter, and Improving the ways people work together, are trademarks or registered trademarks of TogetherSoft Corporation. *A "Project Failure" analysis from the Standish Group's "CHAOS" research report series.



JClass ServerChart™

The **ultimate** charting component for **server-side** applications.

Introducing the first truly powerful solution for thin-client data visualization: JClass ServerChart. With a wide range of chart and graph types and built-in image map tag generation, JClass ServerChart gives end users of your browser-based apps access to business-critical data in an intuitive and highly interactive format.

- multiple business and scientific chart types
- integrates with all major application servers and IDEs
- works in headless environments when used with eTeks' PJA toolkit
- support for Windows 98/NT/2000/ME; Solaris 2.8, Redhat Linux 6.2, AIX 4.3.3
- support for JDK 1.1.8, 1.2.x, 1.3.x

sitraka
software
the Java™ advantage

Download a free eval now

www.sitraka.com/serverchart1/jp

1 888-361-3264 | Europe +31(0)20 510 67 00

© Copyright 2001 by Sitraka Inc. All rights reserved. Sitraka, the Sitraka logo and JClass are trademarks of Sitraka Inc. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All other product names are trademarks or registered trademarks of their respective companies.

©2001 BEA Systems, Inc. BEA and WebLogic are registered trademarks and How business becomes e-business is a trademark of BEA Systems, Inc.

With BEA WebLogic® you'll develop applications faster. So you can really get down to business.

The world's #1 e-business platform just got a lot more powerful. BEA WebLogic now has personalization and portals, application integration, customer management, e-commerce, B2B integration and business process management.

So now you can develop and integrate your applications in weeks instead of months.

Which means you might find that you'll have some extra time on your hands. How you spend it is, well, up to you.

Download your free trial copy of BEA WebLogic today at www.bea.com/download. And be sure to check out all of our free developer resources at **developer.bea.com.**

bea™

How business becomes e-business.™

JAVAPro

COVER STORY

Automate SOAP Calls in Java with the Proxy Pattern

by Henry Bequet

Having trouble integrating your Web apps? Learn how SOAP, when combined with Java and the Proxy pattern, can be a powerful tool that solves communication problems between remote computers that need to exchange objects over the Web.

SERVER

24 Discover Your Inner Classes

by Harris W. Kirk

Writing all those unit tests to verify the proper operation of code can be a real pain. And the more tests you write, the more your code becomes unmanageable. JUnit is a testing tool that was designed to make it easier for Java programmers to unit test. See how this tool works with inner classes to provide a simple, scalable testing framework that keeps your code manageable.

33 COLUMN: JAVATECTURE
Java: Closer to .Net than You Think

by James Cooper

The .Net languages bear a strong resemblance to Java. In fact, you can write almost exactly the same code in both systems. Is Microsoft trying to tempt Java programmers to come on over? Learn some subtle differences and decide for yourself where .Net is heading.

WEB

37 The Lowdown on Uploads

by Alexandre Calsavara

File uploading is an integral part of many Web applications, but the Java servlet specification doesn't provide any support for it. Fortunately, the servlet API provides the low-level control needed to develop a reusable, generic Java component to handle file uploading.

53 COLUMN: WEBLICATION
Sophisticated Databases

by Peter Varhol

JavaServer Pages let Java programmers create Web-based user interfaces for manipulating data in databases. Once you have retrieved the data from your database, you can display that data to the user. Learn how good user-interface design and coding practices help you get the best performance from your database calls.

90

```
public class QuoteClient {
  URLEndpoint service;
  static SOAPMessage cannedQuery;

  static {
    try {
      MessageFactory factory =
        MessageFactory.newInstance();
      SOAPBody body;
      SOAPEnvelope envelope;
```

Contents Continued

BPA INTERNATIONAL
PRINTED IN USA

Java Pro is published monthly—plus a special issue in September—for $34.97 per year, by Fawcette Technical Publications Inc., 209 Hamilton Avenue, Palo Alto, CA, USA, 94301-2500. Tel. 650-833-7100; Fax. 650-321-3818. **Postmaster:** Send address changes to *Java Pro*, P.O. Box 54584, Boulder, CO 80322-4584. Application to Mail at Periodicals Postage Rates is Pending at Palo Alto, CA and at additional mailing offices. **© 2002 Fawcette Technical Publications Inc.**, all rights reserved. *Java Pro* is an independent publication not affiliated with Sun Microsystems. Sun Microsystems is not responsible in any way for the editorial policy or other contents of the publication. All contents of Java Pro are copyright © 2001 by Fawcette Technical Publications Inc., unless otherwise noted. "VBITS" and "Interactive Developer" are trademarks of Fawcette Technical Publications Inc., a California Corporation, James E. Fawcette, President. Java is a trademark of Sun Microsystems. Rather than put a trademark symbol in every occurrence of other trademarked names, we state that we are using the names only in an editorial fashion with no intention of infringement of the trademark. Although all reasonable attempts are made to ensure accuracy, the publisher does not assume any liability for errors or omissions anywhere in the publication. **Canadian GST#: 86831 0764 ISSN: 1096-4495.**

BEA eworld

THE 7TH ANNUAL BEA DEVELOPER CONFERENCE
San Diego, CA February 24–27, 2002

bea

4000 Developers
130 Conference Sessions
100 Partner Exhibits

3 days in San Diego.
Application infrastructure
'til the sun burns out.

Alfred S. Chuang
Founder, President, and CEO, BEA

John E. Davies
Vice President, Architecture Group, Intel

Scott Dietzen, Ph.D.
CTO, BEA

Ed Roman
CEO, The Middleware Company

Adam Bosworth
Vice President, Engineering, BEA

San Diego will boast a lot more than great weather this February. It'll be the best place on earth to learn about the latest developments in e-business software. BEA eWorld 2002 will feature a unique combination of hands-on demos, deep technical education, networking opportunities, and keynote presentations by some of the sharpest minds in IT. You'll get a front-row view of the future of e-business.

REGISTER NOW! www.bea.com/events/eworld/2002/ or call (404)240-5506. Hurry! Space is limited.

[SAVE $200 WHEN YOU REGISTER BEFORE DECEMBER 14, 2001!]

© 2001 BEA Systems, Inc. All rights reserved. BEA is a registered trademark of BEA Systems, Inc. All other company and product names are trademarks of their respective owners.

Platinum Sponsors:

intel. Sun microsystems

Gold Sponsors:

accenture INTERWOVEN PWC CONSULTING VIGNETTE

JAVAPro

GROUP PUBLISHER, ENTERPRISE GROUP, Jeannine Barnard

EDITORIAL
EDITORIAL DIRECTOR, Dan Ruby
EDITOR, Kay Keppler
MANAGING EDITOR, Janaya Reitz
TECHNICAL EDITOR, Daniel F. Savarese
ASSISTANT EDITOR, Charles Ferrara

CONTRIBUTORS
James W. Cooper, Claude Duguay, Rick Grehan, Andy Patrizio, Daniel F. Savarese, Peter Varhol

ART & PRODUCTION
VICE PRESIDENT, ART & PRODUCTION, Michael Hollister
SENIOR ART DIRECTOR, Bruce Gardner
PRODUCTION DIRECTOR, Christine Marquez
ART DIRECTOR, Ginger Cross
ASSOCIATE PRODUCTION MANAGER, Kathleen Cygnarowicz
SENIOR DESIGNER, Kim Le
ASSOCIATE ONLINE ART DIRECTOR, Brian Rogers
ASSOCIATE WEB PRODUCER, Shane Lee
ASSOCIATE PRODUCER, Mona Bedi

ADVERTISING SALES
ASSOCIATE PUBLISHER, Shana Deane
PUBLISHER, FTP ADVERTISING NETWORK, Paul Spyksma
INSIDE SALES MANAGER, David Seymour
EASTERN REGIONAL SALES MANAGER, Brian Levine
SOUTHERN REGIONAL SALES MANAGER, Chris Kourtoglou
NORTHWESTERN REGIONAL SALES MANAGER, Janice Mahler
PRODUCTION COORDINATOR, Tami Stoeckle

CIRCULATION
CIRCULATION MANAGER, Meg Clark
CIRCULATION ASSOCIATE, Gerardo Guzman
CIRCULATION ASSOCIATE, Joyce Hoelzle

MARKETING
SENIOR VICE PRESIDENT, Rob Tobias
MARKETING COMMUNICATIONS MANAGER, Libby Dickinson
MARKETING DESIGNER, Margaret Horoszko
MARKETING SPECIALIST, Susan Ogren
DESIGNER, Helga Taylor

CONFERENCES
VICE PRESIDENT, Tena Carter
GENERAL MANAGER, Tim Smith
ASSOCIATE EDITOR, Maryam Ghaemmaghami
CONFERENCE MARKETING MANAGER, Jennifer Brucker
CONFERENCE MARKETING DIRECTOR, Thea Schwartz
CONFERENCE MARKETING PLANNER, Shannon Sullivan
EXHIBIT SALES MANAGER, Tina Fontenot
CONFERENCE SALES PLANNER, WINDOWS GROUP, Heidi Corcoran
WESTERN REGIONAL SALES MANAGER, WINDOWS GROUP, Tanya Cecchin
CONFERENCE OPERATIONS PLANNER, Leslie Hasvold
CONFERENCE ASSOCIATE WEB PRODUCER, Lyndon Lloyd
EASTERN REGIONAL SALES MANAGER, WEB CONFERENCES, Larry Westfall
SPONSORSHIP COORDINATOR, WEB, Jessica Bartlett

CUSTOMER SERVICE
CUSTOMER SERVICE REPRESENTATIVE, Laurie Groenewold
CUSTOMER SERVICE REPRESENTATIVE, Nicholas Cadigan

OPERATIONS
CHIEF FINANCIAL OFFICER, John Sutton
DIRECTOR OF HUMAN RESOURCES AND FACILITIES, Philip Ziman
SYSTEM ADMINISTRATOR, Tin Cao
INFORMATION SYSTEM SPECIALIST, Todd Gallagher
ACCOUNTING OPERATIONS MANAGER, Linda Flowers
DIRECTOR OF FINANCE, Darlyn Philips
A/P ACCOUNTANT, Elena Ostrovsky
PAYROLL SPECIALIST, Linda Darmas
STAFF ACCOUNTANT, Iain Neillands
CASH MANAGEMENT ACCOUNTANT, Betty Wu
COLLECTIONS ANALYST/CASH MANAGEMENT ACCOUNTANT, Lina Yonemitsu
FINANCIAL ANALYST, Neema Buruku
HR MANAGER, Pamela Davis
FACILITIES SUPPORT, Staci Matlack

FAWCETTE TECHNICAL PUBLICATIONS
PRESIDENT, James E. Fawcette
CHIEF OPERATIONS OFFICER, Patricia Cameron-Loyd
VICE PRESIDENT, CHIEF INFORMATION OFFICER, Aaron Weule
VICE PRESIDENT, INTERNET PRODUCTS, Matt Carter
GROUP PUBLISHER, Lynne Matthes
CORPORATE COUNSEL, Wilson, Sonsini, Goodrich & Rosati

CLIENT

56 Don't Get Stuck in the GUI Thread
by Charles W. Kann
Teaching concurrent programming requires that students understand the importance—and proper use—of threads. Caution must be exercised, because failure to consider the GUI thread can shut down the interface. See how you can avoid these problems.

63 COLUMN: VISUAL COMPONENTS
Make Progress with JSequence
by Claude Duguay
Installation and diagnostic procedures and automated updates are situations in which programmers need to understand the progress of a sequence of events. Find out about the component JSequence, which provides an infrastructure that shows users sequential progress as a list view, so they can see what's been done at any time.

MOBILE

69 How to Climb a B-tree
by Rick Grehan
You can extend the capabilities of the db4o database system by defining a simple linear-array index and storing that index as an object in the database. Using the popular B-tree indexing structure allows you to save considerably on disk access time (and memory space). Discover the benefits of this lightweight, portable solution.

78 COLUMN: JAVA TO GO
Cure Your Waba Woes with a Serial Socket
by Rick Grehan
The Waba SDK comes with example applications, including a Web application. Although you may have no trouble getting the app to run on your desktop, you may have difficulty getting it to run on the Palm. Learn how to develop a homegrown solution to this problem.

DEPARTMENTS

10 EDITOR'S NOTE
by Kay Keppler

12 LETTERS

18 IN BRIEF

93 AD INDEX

90 PRO SHOP
by Daniel F. Savarese
The Java API for XML Messaging (JAXM) has just been released. Although it is only one of many components for building Web services, it just may prove to be the most crucial. Use it to facilitate the sending and receiving of XML messages to and from Web services.

REVIEWS

16 SAVAJE XE
A Java SE environment for handhelds

96 BOOK REVIEWS

Don't just get ahead of the game

(dominate it)

It's tough being a developer – you're expected to be part fortune teller, part troubleshooter, part super hero – trying to stay ahead of technology and still deliver the best apps possible, on time and on budget.

And when you think you've got everything under control, along comes Web Services. Time to start again? Wrong. SilverStream eXtend™ makes creating, assembling and deploying Web Services enabled J2EE apps easy – it runs on all industry leading J2EE application servers and won't compromise on quality. To put it simply – it's the best Web Services platform in the World.

But don't take our word for it. Visit www.silverstream.com/dominate to download a free 30 day trial now or call 1-888-823-9700 (US) or 1-978-262-3000 (outside the US).

SilverStream
eXtend™

www.silverstream.com

JAVAPro Online

www.javapro.com

HOW TO REACH US

Editorial Offices: Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, CA 94301-2500; Phone: 650-833-7100; Fax: 650-853-0230; Web: www.javapro.com; e-mail: java-pro@fawcette.com.

Article Submission: To submit articles for publication please contact Kay Keppler, Editor, kkeppler@fawcette.com. Download the Author Guidelines at www.javapro.com/authguid.asp.

Product Reviews: To submit products for review please contact Kay Keppler, Editor, kkeppler@fawcette.com.

Product Announcements: Please e-mail press releases to pressreleases@fawcette.com.

Reprints and Permissions: For all *Java Pro* editorial and advertising reprints contact PARS International Corp., 114 East 32nd Street, Suite 503, New York, NY 10016; Phone: 212-221-9595; Fax: 212-221-9195.

To Quote from an Article: Please contact Tami Stoeckle, 209 Hamilton Avenue, Palo Alto, CA 94301-2500; Phone: 650-833-7118; E-mail: tstoeckle@fawcette.com. Specify the issue date and title of the article, the portion you would like to quote, and the purpose.

Photocopy Rights: Permission to photocopy for internal or personal use may be granted by Fawcette Technical Publications. Please contact Tami Stoeckle for more information.

Customer Service and Subscription Information: For subscription orders, inquiries, or address changes please contact Customer Service, *Java Pro*, P.O. Box 54584, Boulder, CO 80322-4584; Phone: 303-945-5282; Fax: 650-321-3818. Foreign and Canadian orders must be payable in U.S. dollars, plus postage. The surface rate to Canada and Mexico is $52.97 per year. For all other countries the air mail rate is $78.97 per year.

Back Issues: To order *Java Pro* back issues, call 650-833-7100 or 800-848-5523 and ask for Customer Service. Back issues cost $10. Additional postage will be charged to deliveries outside the USA.

Visit *Java Pro* online for this extended content and more:

ONLINE CODE

JP0201HB

Develop Scalable Apps with SOAP

by Henry Bequet

SOAP solves communication problems between remote computers. Download code that shows how, using SOAP and the Java proxy pattern, you can develop a powerful, low-cost B2B app that doesn't require an application server.

JP0201AC

Add File Upload Capabilities to Your Web Site

by Alexandre Calsavara

File upload can make many tasks easier, such as archiving files, submitting large amounts of data, and maintaining your Web site. Download code that helps you create a reusable, generic Java component specifically designed to handle file uploads.

JP0201RG

Build an Index for Your Handheld Database System

by Rick Grehan

The db4o database system is lightweight and perfect for handhelds. Download code and create an index (modeled after the handy B-tree data structure) that will save you considerably on disk access time and memory space.

JP0201VC

Measure Your Users' Progress

by Claude Duguay

Download code that you can use to create a simple component that shows users their progress through items in a sequence.

JP0201HK

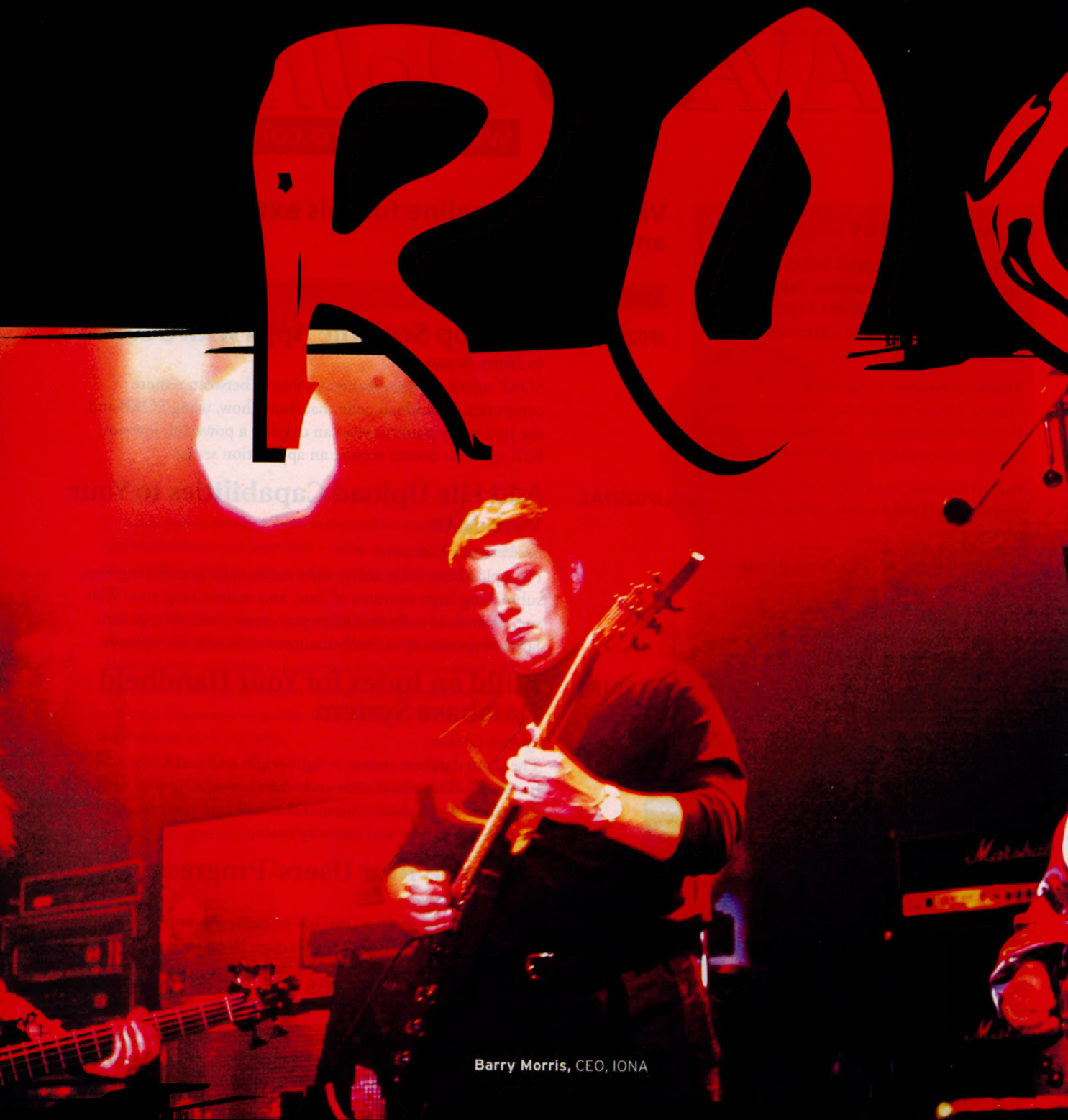
Use Inner Classes for Unit Testing

by Harris W. Kirk

Unit testing can be painful, but using inner classes in your unit-testing regime can make your life just a little easier. Download test cases and try it out for yourself.

JAVA NEWSLETTER SIGN-UP

javapro.com Every week, the Java e-mail newsletter brings you up-to-date news, technical information, opinions, interviews, and analysis. Customize the newsletter to suit your unique professional needs by setting preferences for topics and technologies such as J2EE, middleware, and application servers; JME, devices, and embedded development; data access; servlets and JSP; Web Services; and/or XML. Sign up for *free* at www.javapro.com.



Barry Morris, CEO, IONA

Java developers and rock guitarists like to build their creative licks and flourishes on a rock-solid infrastructure. The guitarist has drums and bass; the Java developer has the Orbix E2A Application Server Platform™.

IONA Will Make You a Star

We'll Make You Faster
We'll Make You More Productive
We'll Expose Your Applications as Web Services
We'll Connect You to a World of Applications
We'll Save You a Load of Money, Too

Orbix E2A | APPLICATION SERVER PLATFORM™
Fully J2EE™ Certified

Grab yourself a free download from **www.iona.com/downloadJ2EE** and cure your Java development blues.

IONA® | E2A™

Copyright 2001 © IONA Technologies PLC. E2A and Orbix E2A Application Server Platform are trademarks of IONA Technologies PLC. Orbix is a registered trademark of IONA Technologies PLC. All other product and service names are the trademarks and registered trademarks of their respective owners.

Rushing to Judgment

by Kay **KEPPLER**

These days everybody is stampeding to Web services. In the last week, at least a half-dozen software vendors have e-mailed me, announcing major Web services initiatives. New products, product revisions, software partnerships, consulting services—they're all coming together for companies who hope to turn their existing business assets into a Web model that can earn cold, hard cash in the economic downturn.

The rush to this Internet business strategy seemingly has been led by Microsoft, which early on embraced XML and SOAP and extended these to its .Net/Hailstorm platform. Sun, which is adopting an open software standards approach to its Web services strategy, has taken a different road that seems somewhat more circuitous. Lest you think Sun has lagged behind Microsoft's initiatives, I'm here to tell you that Scott McNealy says it isn't so. In fact, Sun has been porting TCP/IP to Unix since 1982. Sun is just more backward in its marketing. So I'm glad we've got that cleared up.

> Java developers are unlikely to see Microsoft's Java permutation as useful, and will rightly see .Net as .not open, .not flexible, and .not Java

One of the software developments that Microsoft has announced for .Net is Visual J#, which should be ready by mid-2002. J# is the Java that isn't Java; the Java that you write once and run only on Windows. I could be wrong, but I thought we'd seen this once before, only back then Microsoft called the language Visual J++, and Sun got really mad about it.

The Visual J++ thing got settled about a year ago—Sun won the legal argument that Microsoft violated the terms of its licensing agreement by altering the language for the Windows platform, and Microsoft therefore can't use the updated Java binaries in Visual J++. Now we have J#. Do you think Sun's mad about that? My guess is, those lawyers that beat Microsoft the last time aren't sitting around drinking double lattes and watching the world go by.

Microsoft thinks it can get around the licensing problems by using Java language syntax, not the underlying runtime environment. Developers will not be able to use J# to build standalone Java applications, but can use the Java language to write code that works only with .Net. By limiting J#'s capabilities in this way, Microsoft seems to hope to avoid another lawsuit.

The Redmond giant came down the Java road once before and got bashed up a bit in the process. Perhaps the narrow capabilities of J#—using it to migrate Java programs to .Net, or perhaps, more explicitly, J++ programs to .Net—will be suitable for those developers already using Microsoft technologies. Where that leaves C#—the Java-like language Microsoft developed to create a migratory pathway for .Net—is another question.

Java developers are unlikely to see Microsoft's Java permutation as useful, and will rightly see .Net as .not open, .not flexible, and .not Java. Industry analysts, including Mark Driver for Gartner and Chris Dial for Forrester Research, have said that Microsoft will pitch J# to IT managers, because most Java platform developers will ignore it.

The success or failure of J# is relevant primarily in the larger discussion of where Web services is headed. The Microsoft forces have maybe done an end run with their Java knockoff. But Sun ONE—Sun's open source strategy on Web services—brings flexibility to its table. With the release of the Java APIs for XML parsing and processing, plus a bunch of recent product revisions, the components for building Web services in Java are finally coming together.

It's all a contest to lure developers and the managers who make buying decisions in times when resources are tight and companies want to make money from their existing information assets. Microsoft plans to get them with its tightly integrated, wholly proprietary platform—never mind that Gartner says that 60% of Windows applications will have to be rewritten to run on .Net. Sun plans to lure customers with its open standards and integratable architecture. Who will win? I'll tell you what Scott McNealy thinks. "It's really mankind versus .Net," he says. "And mankind is going to win." JP

Kay Keppler

by Kay Keppler, Editor
kkeppler@fawcette.com

Ride the next WAVE

Beyond JMS & Messaging.

The "next wave" in enterprise messaging will not be judged by speed alone. The "next wave" of Web services will push distributed systems to the limits of scalability, reliability, and flexibility.

SpiritSoft goes beyond JMS to offer the only open enterprise messaging-based technologies that deliver multi-plug messaging, multi-channel delivery and dynamic event management.

Find out more about the "next wave" JMS architecture by downloading our whitepaper.

spiritsoft™
the integration company

Download our white paper
"The SpiritSoft Architecture: Beyond JMS & Messaging"
Visit **www.spiritsoft.net/downloads**

Letters to the Editor

Letters to Java Pro *are welcome; they may be edited for form, fit, and style. Please send them to Letters to the Editor,* Java Pro, *c/o Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, CA 94301-2500; fax them to 650-853-0230; or e-mail them to java-pro@fawcette.com.*

Unconventional Data Doings

I always enjoy James Cooper's Javatecture column—he writes well and chooses interesting topics. About his "Courage in Profiles" (November 2001): heresy! If you are putting data on the file system, it should always be stored in XML or an RDBMS! Just kidding.

In an earlier life, writing DOS and Windows programs with C, I used to append such data to the executable (yes, really!). The same trick works with Java, I believe. You tack your data on the end of the class file, and the JVM does not complain. Of course, every time you recompile, the data must be tacked on again. This approach can be useful if the data is not supposed to be visible and edited by users (for example, if the software provider/vendor is the only one allowed to edit the data).

Of course, it is trivial to concatenate data to the class file. For example, to tack on the source code to a class file, do this in Windows:

```
copy Main.class+Main.java
```

When you peek at Main.class with a hex editor, you will see the source at the end.

Another approach is to set up a data array (or String) in the program with a "marker" in the array such as "~~". Now you can edit this section of the class file (either manually or with a program) and insert whatever data you wish. When the program is run, it uses the data you entered. This is weird stuff, but I confess to having done it, and others have too.

Stan Jordan

James Cooper responds: Thanks for your somewhat unconventional idea. You can certainly do that, but it doesn't seem particularly object-oriented to tack data onto a class. It does provide some secrecy, however, if you don't want people to figure out what you are doing.

XML: Better than Ini

At the end of the Javatecture column "Courage in Profiles," (November 2001) James Cooper asks for ideas on how to deal with configuration data.

Ini files are simple, but there is a better way to store the same information—XML. I simply extend Properties and add methods to serialize/deserialize a Properties object to/from XML file using SAX. It is easier than it sounds. The best part is that there is no need to write a parser. Java comes with an excellent one already.

Many tools are available for managing XML (including Notepad) as well as for checking the syntax.

At one point, I had to write a configuration server, which serves XML back to applications that are starting up—but that could be material for a whole article. Same concept, just XML comes over the network from a centralized repository, not from a local file.

Julian Chultarsky

James Cooper responds: You are absolutely right about XML, and it is the subject of December's column "Caught in a Lobster Trap."

Further Testing

Thank you for writing and publishing "Take the Work out of Unit Testing" (October 2001). I found it a useful and informative read. I would like to point out one correction. The article states: "The philosophy behind JUnit, and unit testing in general, is that shortly after you write a new class method, you should write a test for that method." This may be the philosophy behind unit testing in general, but JUnit's (and Extreme Programming's) philosophy is that you write the test first. I applaud Daniel Savarese's self-discipline in writing the tests after implementation. This is when many folks fall back on the arguments against writing unit tests pointed out in the article. Writing the test first helps save us from ourselves and helps us better understand what we need to implement.

Erik Meade

Isolationist Theory

I just finished scanning "The Good Side of Isolation" by Alexandre Calsavara (November 2001). Being the reluctant architect on a reasonably large public-facing e-business site developed using J2EE technologies (including JSP), the topic the article intended to address is of great interest to me. Originally the site was built in collaboration with a respected Web-development firm, which provided much of the creative content and user interface for the site. I was unable to persuade that firm's development team of the benefits to be gained by separating presentation generation from interaction control. The unfortunate result was the delivery of JavaServer Pages with hundreds of lines of Java code. As my team moves forward with plans to implement the next generation of this application, I have been spending a considerable amount of time examining other technology choices.

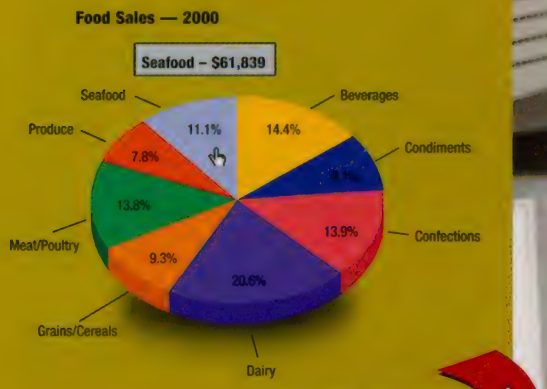
Fundamentally, the problem to be solved as identified in the article is "isolating the page design from the application code." When you choose to embed HTML in Java code or Java code in HTML markup, you have failed to solve this fundamental problem. This is the downfall of JSP as well as the available templating engines. And it is the source of my disappointment with this article. Despite his claims to have solved the separation problem, the author's solution requires embedding non-HTML tags in HTML markup. In essence he's written his own very limited templating engine. If that is what one wants to do, open source implementations such as the Jakarta project's Velocity are a much better choice.

Even better, however, are techniques that truly solve the separation problem with requiring the addition of non-HTML elements in the page design. Consider

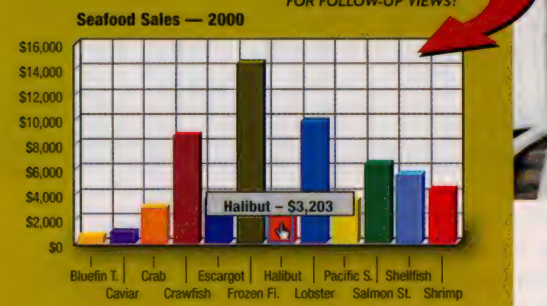
Continued on page 16.

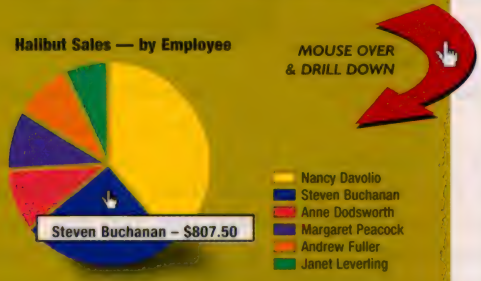
Tired of waiting? Get the fastest Java charting tool on the planet—PopChart™ from CORDA.

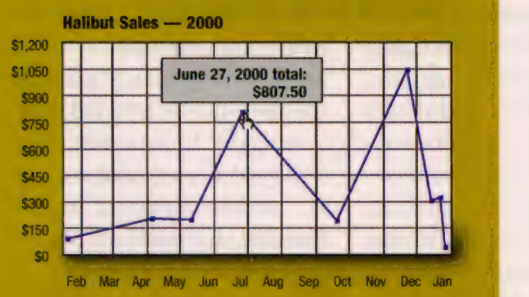
AS DATA CHANGES, UPDATES ARE IMMEDIATE THROUGH EVERY LEVEL!

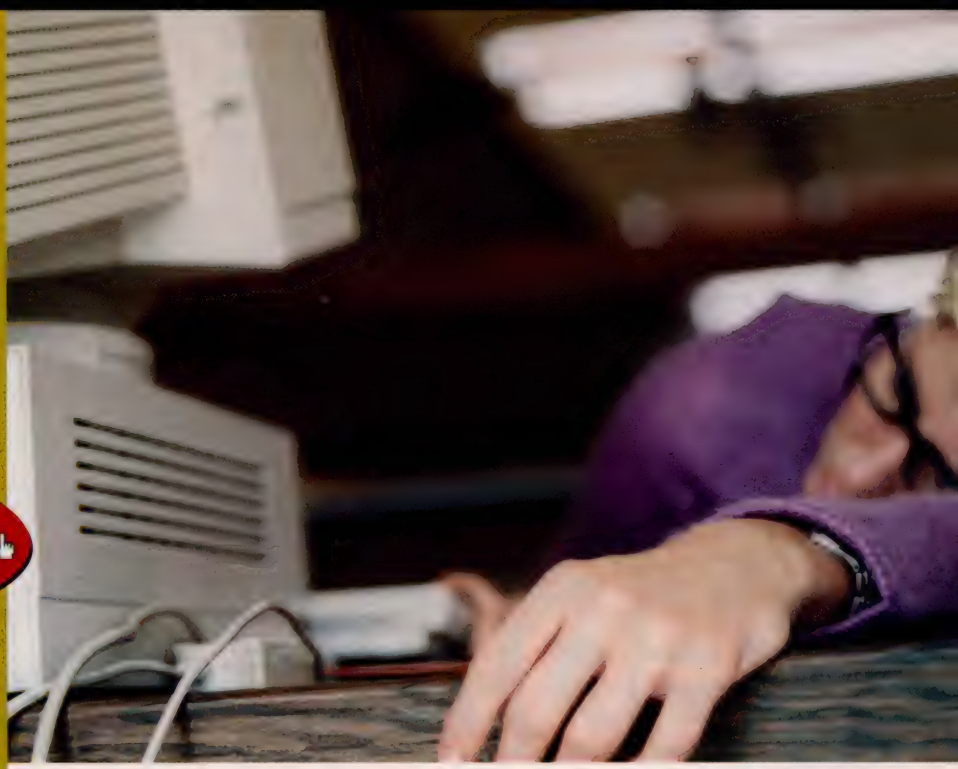


MOUSE OVER & DRILL DOWN FOR FOLLOW-UP VIEWS!









How do the most highly-trafficked web sites generate dynamic, interactive, data-driven charts and graphs?

With PopChart™ from CORDA Technologies.

- **Users don't wait.** PopChart is up to five times faster than other charting solutions.
- **Platform independent.** 100% Java works with all web app servers and browsers.
- **Easy set up.** Deploy on any system in minutes with built in Java API.
- **Support for wireless devices.** Generate images for pocket PCs, PDAs, and WAP cell phones.
- **Highest-quality charts and graphs possible.** PopChart auto detects browser capability, then delivers SVG, FLASH, GIF, or PNG image.
- **More information.** PopChart has drilldown and pop-up text for accessing instant additional information.
- **508 Compliant.** To meet federal government requirements, PopChart automatically generates descriptive text that makes the charts and graphs accessible to the visually impaired.

Visit our site, or call us at 801-805-9400. Your data has never looked better—nor has it been so useful.

CORDA

Interactive Data-Driven Graphics

www.corda.com

SavaJe XE

A small environment

by Rick Grehan

The trend to put Java into the palm of your hand—literally—continues. More specifically, the balancing act continues; the balancing act that seeks to provide a Java environment that is simultaneously small enough to leave reasonable room on the handheld, but powerful enough to be useful. The latest juggler on this stage is SavaJe XE (yes, it is pronounced like "savage").

SavaJe XE is not simply a Java derivative for handhelds; it is an entire Java Standard Edition (SE) environment for handhelds. (This reveals the secret of SavaJe's name: It is an anagram for "Java SE.") To be accurate, SavaJe is a 32-bit, multithreaded operating system that implements a JDK 1.3 runtime environment, includes a JVM, and provides the APIs of the Java SE. Although currently, the product runs only on the Compaq iPAQ and Psion Netbook handhelds, SavaJe is targeted at information appliances, set-top boxes, smartphones, and so on. Written entirely in C, SavaJe's current processor platform is StrongArm. SavaJe's documentation says that a 190-MHz StrongArm is required, along with 12 MB of non-volatile memory and 32 MB of RAM. (A platform porting kit should be out by the time you read this.)

The SavaJe engineers have worked to make the OS adaptable to hardware nuances. For example, SavaJe can run on screen dimensions ranging from 240x320 up to 1024x768, and supports 8-, 12-, and 16-bit color systems. SavaJe also supports PC Cards. It can even handle a keyboard directly connected to the StrongArm, as well as a serial keyboard using a COM port. SavaJe's device drivers are "Unix-like," providing modular integration with the OS. Device drivers can be constructed in Java, C, or ARM assembler (the latter form of drivers can be built using GNU tools). Codecs (for example, MP3, AVI, voice recognition, and so on) are added to the OS in a similar fashion. A SavaJe developer kit that facilitates the building of device drivers is slated to be available in the first half of 2002.

Quick FACTS SavaJe XE puts lots of capability into a small space, but you can't simply migrate a desktop application to a handheld without running into some problems.

The SaveJe engineers have also worked to make SavaJe as comfortable to Java programmers as possible. As already stated, SavaJe runs an SE Java environment (and can also run PersonalJava applications). The SavaJe runtime environment supports AWT and Swing (and the SavaJe engineers have told me that "nothing unusual" had to be done to adapt AWT and Swing to the small screen).

SavaJe Inside

I tested an early version of SavaJe (version 0.1.2) on a Compaq iPAQ (see Figure 1). Unfortunately, I was given the iPAQ with SavaJe preloaded, and no hardware to transfer files from PC to iPAQ, so I had no way of downloading new files for testing on the device. Nevertheless, the iPAQ was preloaded with applications that were grouped in three sections: applications, demos, and tools. The applications included:

- Picture: This is for viewing graphics files. I was able to open and view the GIF files that accompanied the Web pages that were loaded on the iPAQ.
- MP3: This is a music-file player. Unfortunately, no MP3 files were included on the device, so I couldn't test it.
- Assistant 1.5: From Espial group, this application is a calendar, to-do list, and address book.
- Editor: This simple text editor is not much more (actually, a good deal less) than a handheld version of Notepad.
- Escape: This is a Web browser from Espial Group. It works reasonably well with the preloaded Web pages (that did include graphics). It has the interesting capability of being able to render pages either so that they fit the handheld's screen, or so that the handheld's screen becomes a "virtual window" into a full-sized rendition of the page. (Sometimes, neither way works well.)

The demo applications provided with the version of SavaJe I tested were apparently chosen to illustrate the system's graphics capabilities. The demonstrations include the obligatory Tetris-like game (called "Blocks"), an application that bounces shaded balls around the screen, another application that rotates multicolored cubes around the screen, and yet another application that moves a combination of shapes, strings, and images (specifically a small, waving Duke) around the screen. Perhaps the most aggressive of the demo applications is the Metalworks e-mail demonstration application.

Figure 1 | SavaJe Environment SavaJe XE can run on the Compaq iPAQ. The environment includes such familiar applications as e-mail, notepad, and calculator.

SavaJe Technologies

11 School Street
North Chelmsford, MA
01863
PHONE: 978-251-2901
E-MAIL: sales@savaje.com
WEB: www.savaje.com
PRICE: Prerelease version, free download from www.savaje.com; Version 1.0, free 30-day evaluation; single copy license for the iPAQ, $100

Metalworks was designed to show off some of the features of Swing. Its SavaJe incarnation on the iPAQ not only accomplishes that, but also reveals some of the issues that SavaJe and its ilk will have to contend with. For example, although Metalworks executes correctly on the iPAQ, merely resizing the new e-mail entry window is a struggle. In addition, the iPAQ is definitely *not* designed for intensive keyboard input. Entering an e-mail note of any size beyond one or two lines is almost unbearably painful.

In short, SavaJe is to be applauded for putting lots of capability in a small space, but you can't simply migrate a desktop application to a handheld.

The view from the top

The possibilities are limitless with JBuilder, the leading Java™ development solution.

Take EJB™ development to the next level with an intuitive visual designer. Rapidly develop and deploy J2EE™ e-business applications to multiple application servers including Borland® Enterprise Server, BEA® WebLogic,® IBM® WebSphere,® and iPlanet™ Application Server. Develop and deploy applications on Windows,® Linux,® Solaris,™ and Mac® OS platforms. Efficiently collaborate as a team with support for leading version control systems. Build data-driven Web applications using servlets, JSP,™ and XML.

View code in a dramatic new way with UML code visualization. Experience extreme productivity with refactoring, unit testing, and documentation tools.

See for yourself at www.borland.com/new/jb6/5002.html

Borland®

Editors' Choice JavaWorld June 2001 WINNER

Java Magazin EDITOR'S CHOICE BESTES PRODUKT 2001

eWEEK excellence AWARDS

VSJ READER AWARD 2001

PRODUCT EXCELLENCE AWARD 11th Annual

JAVAPro 2001 READER'S CHOICE WINNER

Made in Borland® Copyright © 2001 Borland Software Corporation. All rights reserved. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All other marks are the property of their respective owners. • 12429.2

Significant user interface concerns have to be considered.

The tools include a network configuration tool for setting things like IP address, subnet mask, default gateway, and so on. A calibration application is provided for calibrating the stylus (which seems to want to go out of calibration at random times). The application also includes a tool whose sole purpose, it seems, is to kill applications that can not otherwise be stopped. (I found this to be common, and attributed it to the early version of the product. Some applications defy being killed. There are no menu items or exit buttons for terminating the application, and most applications simply ignore the buttons on the iPAQ's front panel. Even then, the "Java process killer" application isn't sure-fire. I had to kill some applications more than once before they stayed dead.)

I was able to do most of my exploring in the system through another tool, the Java shell application: jsh. This application opens a prompt in a text console window. I can launch the iPAQ's "soft" keyboard, and type in a host of Unix-style commands to list directories, navigate through the file system, create new subdirectories, list active processes, report the number of filesystem blocks still available, and so on. (Through jsh, I discovered that the RAM filesystem had about 10 MB of free space available.) Happily, the jsh output responded to the iPAQ's directional pad, so I could scroll back through output that had disappeared off the top of the screen.

Technology to Watch

The prerelease version ("open beta") of SavaJe is available for free from the SavaJe Web site at www.savaje.com. The version 1.0 release (which should be available by the time you read this) is also free, but as a 30-day evaluation. A single copy license for the iPAQ will be $100.

This product holds a great deal of promise. It's obvious that the SavaJe engineers focused on connectivity. Ethernet support for some CF cards is provided, as is access to the serial port through the javax.comm API. However, some connectivity issues have not yet been addressed—connectivity with IrDA, USB, and BlueTooth, to name a few.

SavaJe is definitely a technology to watch. Once established, it promises a complete SE environment—including features such as RMI, JDBC, CORBA—that, before now, lived only on the desktop. SavaJe already provides a novel, pluggable file system. It currently supports five file systems: a persistent RAM file system, a read-only ROM file system, a FAT-compatible system for CF cards, an MS-compatible network file system, and an experimental read/write file system that attaches to Web servers.

Placing desktop-level weapons in the hands of PDA Java programmers can only make their jobs easier. JP

Rick Grehan is a technical evangelist with Compuware/NuMega. Rick was a senior editor at *BYTE* magazine (where he wrote the JavaTalk column) and at *Computer Design* magazine. He can be reached at **rgrehan@top.monad.net**.

Letters continued from page 12.

XMLC from Enhydra (http://xmlc.enhydra.org). It's another open source project that uses the HTML ID and class attributes and the SPAN tag to identify elements of the page into which application code can insert dynamic content at run time. It's involves absolutely no pollution of the page markup with tags that page design tools do not understand. Developers and page designers agree on the IDs and classes (if necessary) that will tag dynamic content, and each does their part of the job using their own tools.

A similar solution in the making involves the use of XML/XSLT. Although perhaps not ready for prime time, the idea is basically the same—keep the page designers and software developers from forced collaboration on a common artifact. As I understand it, the page designers write/create HTML while the developers deliver dynamic content/data as XML and the two are merged using XSLT. An example of this at work, I think, is the Jakarta Cocoon project. Lutris, the commercial side of Enhydra, is also "experimenting" with such technology (see www.lutris.com/journal/August2000/articles/XMLCandXSL.html). An article on the application of this technology would certainly be appreciated.

Jim Cakalic

Alexandre Calsavara responds: Thanks for your comments and suggestions about my article. However, embedding non-HTML tags in HTML does not mean that you are not isolating page design from application code. It is just fine, as long as whatever you are embedding has only to do with presentation, as is the case with my solution. Even JSP is fine, as long as you restrict the Java code to deal with presentation and leave business logic encapsulated as components. Actually, JSP strongly recommends doing that and has plenty of support for facilitating the use of components.

The problem with the project you mentioned is that the Web development firm simply ignored the most basic concept of good software development: encapsulation. On most unsuccessful projects, the problem is not the tool, but the way you use it.

Going back to my solution, the proof that it works is that plenty of open source, public domain, and commercial solutions use the same or similar concepts, and lots of successful projects are based on these solutions. I never claimed that my solution was original, the only one, or even the best one. In the best spirit of Extreme Programming, it is just the most basic, simple solution that still solves the problem—easy to understand, easy to use, easy to customize, small, clean and, best of all, you don't need a symmetric multiprocessor, multigigabyte RAM, water-cooled server just to generate HTML pages.

Regarding using XML/XSLT, certainly it would solve the problem and achieve the same results I achieved with better encapsulation, and best of all, it is a standard! However, how do you create XSLT using just a standard HTML editor? You don't. XSLT is completely independent of the generated output. It can generate HTML as well as any other format; With XSLT, you don't design pages, but templates that, when applied to XML data, generate something. So, you need a complete new class of tools to work with XSLT. And we must not assume that Web designers who know nothing, or almost nothing about programming, will be able to understand and effectively use XSLT. If your Web masters do understand the technology, go ahead; otherwise you'd better wait until the technology matures.

Rational
the software development company

ENEMIES: Bugs
Changes
Meetings
Changes
Chaos
Changes

FRIENDS: Rational ClearCase

You've got a list. We've got a list. Know what's on our list? Version control. Defect tracking. Parallel development. Support for distributed teams. Stuff to make the stuff on your list go away. Which is why, for three straight years, the people in charge of investing their company's software budget have invested in Rational ClearCase more than in any other Software Configuration Management solution.* Before you make your next investment, check out Beyond Version Control, an objective guide to selecting an SCM solution, at **www.rational.com/offer/scmguide14**.

be liberated

Rational, the Rational logo and Rational ClearCase, among others, are trademarks or registered trademarks of Rational Software Corporation in the United States and in other countries. ALL RIGHTS RESERVED. Made in the U.S.A. ©2001 Rational Software Corporation. *IDC, July, 2001, Software Configuration Management Tools Forecast and Analysis, 2001-2005.

In Brief

Database and Middleware Tools • Developer Tools

Database and Middleware Tools

JDX Combo

This object-relational mapping suite is designed to accelerate the development of object-oriented applications by eliminating the complex task of low-level infrastructure programming in SQL. The suite consists of JDXClassic 3.0, JDXStudio, and JDXServer. JDXClassic 3.0 provides support for complex object modeling, including class hierarchies, allowing integration with legacy data to leverage existing investment. JDXStudio is a Java Swing-based GUI tool for defining/verifying object-relational mapping, forward-engineering database schema, and reverse-engineering Java classes. JDXServer offers object-relational mapping functionality in a separate tier for load balancing and scalability. You can also buy the tools separately. $1,495 for JDXClassic 3.0; $595 for JDXStudio; $595 for JDXServer; $2,495 for the entire suite.

Software Tree Inc.
Phone: 408-557-6769
E-mail: info@softwaretree.com
Web: www.softwaretree.com

Developer Tools

Apycom Java Menu Applets 2.51

This collection of platform-independent lightweight Java applets is designed to assist in the creation of professional Web site menus. The applets provide a quantity of tab sorts, like simple border tabs and Windows-like 3-D tabs. They also allow developers to change colors, fonts, icons, sounds, apply JavaScript calls, and use the applet both with and without frames. Its other features include frame support, user-definable font style, font size, and font face, image icons, mouseover and click sounds, user-defined browser's status string, and toggle mode. $39.95.

Apycom Software
E-mail: order@apycom.com
Web: www.apycom.com

Precise/Indepth for the J2EE Platform

This performance-management tool for server-side Java applications has a performance collector that runs on an applica-

ACCELL/Web

Unify Corp. has brought Java to its ACCELL product line, allowing developers who have written applications in ACCELL/SQL to convert those applications over to Java code with little or no recoding required.

ACCELL/SQL applications are written in Unify's own 4GL language, and developers are able to maintain two separate deployments, the SQL and Java versions, by writing to one code base, the native ACCELL 4GL. Unify says its customers have asked for both SQL and Web versions.

"They've come to us saying we want to Web-enable our existing apps and start to build new applications with Java as the underlying platform," said Dave Glende, vice president and CTO of Unify.

Unify is handling the conversions. Customers bring in their ACCELL/SQL code to Unify's Sacramento, California office and often leave the same day with converted code.

The whole application is converted in the process. The application logic is converted to Java while the forms and interface are converted to JavaServer Pages. Unify's eWave Engine, an embeddable Java application server, is included with ACCELL/Web and acts as the interaction server with the Web server.

There is a minor performance hit from the conversion, but it's not Java's fault, said Glende. "Moving to a browser and a Web server adds a few hops. Performance is slightly slower but still acceptable. It's not really due to Java; it's going through a browser-based interface," said Glende.

ACCELL/Web is available for Solaris and Linux from Unify at www.unify.com, or you can e-mail for information at info@unify.com.

SmartSockets for JMS

Talarian Corp. has released SmartSockets for JMS, a Java Messaging Service-capable version of its message-oriented middleware (MOM) that gives the Sun messaging API many advanced features.

SmartSockets has been on the market for more than a decade as a specialized high-performance messaging solution, offering real-time publish-subscribe, multicast, and message queuing. It's designed for high-volume, fault-tolerant systems, and has been significantly optimized during its 12 years of use.

Although the JMS part is written in Java, the SmartSockets technology is written in C for higher performance and offers fault tolerance, dynamic routing, and monitoring and security—all features not yet in JMS.

By building JMS on top, Talarian gives the relatively new Java messaging service some of the features it does not yet have. "The [JMS] spec does not touch upon standards for many other issues that come into play when deploying production quality software, like security and monitoring," said Steve Gimnicher, vice president of marketing for Talarian.

This is actually the second release of the software; it was originally released

Continued on page 20.

Register and get Conference details at
http://java.sun.com/javaone/

JavaOne℠

Sun's 2002 Worldwide Java Developer Conference℠

BEYOND BOUNDARIES

Moscone Center
San Francisco, CA

JavaOne℠ Conference
March 25–29, 2002

JavaOne℠ Pavilion
March 25–28, 2002
JavaOne Pavilion Preview
March 24, 2002

Experience the power of Java™ technology, the universal services platform.

At the JavaOne℠ conference, learn how Java technology is driving the advances in service delivery. Build and develop with Java technology and discover the freedom to innovate. The future is now.

Register online today using Priority Code JPMG
or call 888-886-8769;
International 781-433-1509

Sponsored by Sun microsystems

Produced by Key3Media Group

Copyright ©2002 Key3Media Events, Inc. All rights reserved. Sun, Sun Microsystems, the Sun Logo, Java, JavaOne, the JavaOne logo, the Coffee Cup logo, the Duke logo, Jini, Forte, and all Java-, Jini-, and Forte-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems Inc. in the U.S. and other countries. Key3Media and Key3Media Events are trademarks or registered trademarks of Key3Media Events Inc. in the U.S. and other countries. Produced by Key3Media, Inc. Sponsored by Sun Microsystems, Inc.

tion server's middle tier, while its user interface uses a thin-client, HTML-based approach allowing developers access to the program from a standard browser, locally or remotely. Data collectors use application instrumentation and Java virtual machine (JVM) sampling while the application discovery component works with a bytecode instrument to provide performance metrics during execution. The program also extracts CPU resource usage data from underlying operating systems to provide a correlated view of CPU usage with Java application components and specific end-user requests. Multiple levels of "method invocations" among Java components are correlated to provide visibility into quantitative relationships among application components. Contact for pricing.

Precise Software Solutions Inc.
Phone: 800-310-4777; 781-461-0700
E-mail: info@precise.com
Web: www.precise.com

Together Control Center 5.5

A development platform designed for teams using Java and Java 2 Enterprise Edition (J2EE) application servers, Together Control Center 5.5 offers a diagram editor, Javadoc code completions, multipane editing, a pattern-builder expert, autocorrecting audits, a DBMS configuration expert, and a servlet pattern generator and editor. Its new features in version 5.5 include support for VB 6, VB.Net, C# (early access), Oracle9i and DB2 7.1 database management systems, and SOAP/Web services support. Contact for pricing.

TogetherSoft LLC
Phone: 919-833-5550
Web: www.togethersoft.com

Continued from page 18.

under the name Workbench for JMS. With this release, performance has been improved and it has been certified to work with J2EE servers. Talarian estimates that the performance gain over standard JMS messaging found in application servers is up to 250 percent faster, depending on the application load.

SmartSockets for JMS is available now from Talarian at a starting price of $55,000. Contact Talarian at www.talarian.com for more information.

Jeode JVM

Windows XP may not have a Java Virtual Machine (JVM), but Pocket PC 2002 will, if BSquare Corp. and Insignia Solutions have anything to do with it. The two are providing a JVM for the new Pocket PC handheld platform, which has a pretty large number of supporters.

BSquare is shipping the Jeode PDA Edition, Insignia's JVM for handheld devices, for Pocket PC 2002. Jeode has passed all of Sun's PersonalJava compatibility test suites and BSquare claims it's the fastest handheld JVM on the market. That's because it compiles as much code as possible, rather than interpret it line by line, as other embedded JVMs do.

By compiling some of the code, the Jeode JVM can be anywhere from six to ten times as fast as other JVMs, according to Denise Slocombe, director of business development for BSquare. Performance is necessary as handhelds do more powerful tasks.

"Pocket PCs are now being deployed with enterprise applications. With the ability to add on a phone they become a wireless device. So the ability to connect to the Internet and process Java applications makes the Pocket PC an even more powerful tool," said Slocombe.

Microsoft has a number of supporters for Pocket PC, including Casio Computer Corp., Cesscom, Compal Electronics Inc., Compaq Computer Corp., Hewlett-Packard, and Toshiba Corp. Thus far, Compaq is the only vendor to provide the Jeode JVM with its handheld, the iPaq H3800. For more information, check out BSquare's Web site at www.bsquare.com.

Oak Grove Reactor 5 BPM

Oak Grove Systems announced that its Oak Grove Reactor 5 business process management (BPM) engine will be able to bridge process integration between Java 2 Enterprise Edition (J2EE) applications and Microsoft .Net applications. Applications written in either Java or .Net languages can now trigger a Reactor 5 business process and can participate in business processes managed by Reactor 5 via the SOAP Web Services protocol. Oak Grove Systems can be reached at www.oakgrovesystems.com.

Kada Mobile Platform and VoIP Phones

Cisco Systems has chosen Kada Systems' Kada Mobile Platform to support Java applications in Cisco's family of Voice over Internet Protocol (VoIP) phones. Cisco chose Kada because Kada not only has a small footprint, but also is incremental in voice-enabling previous generations of Java phones not meant to carry VoIP, according to Jim Acquaviva, CEO of Kada Systems. Cisco has not yet decided on whether it will open its phones to third-party developers. For more information visit Kada at www.kadasystems.com.

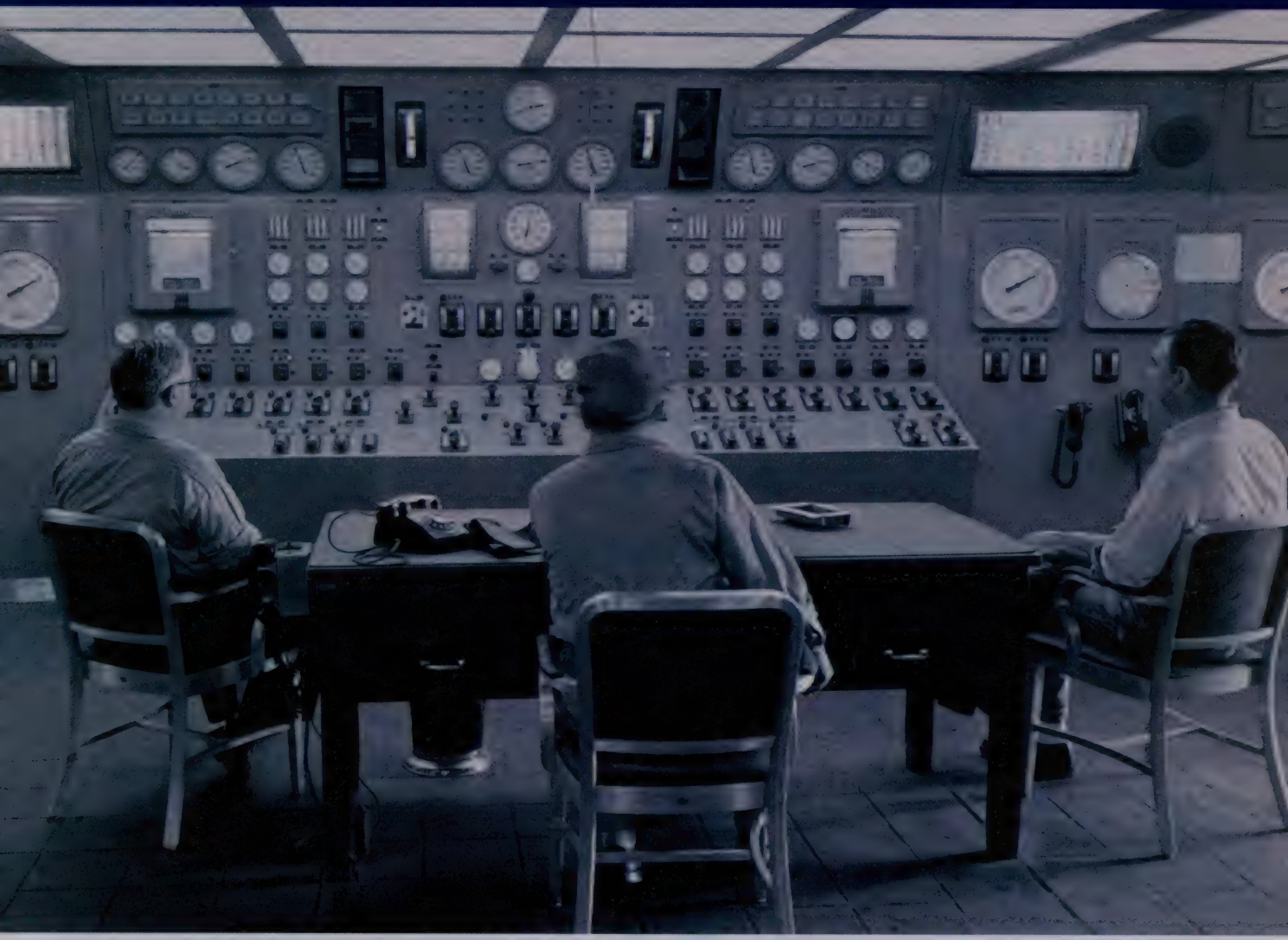
Interstage Application Server 4.0

Fujitsu Software Corp. is the latest application server vendor to gain Sun's blessing. Its Interstage Application Server 4.0 has completed Sun's J2EE 1.2 compatibility tests for Windows, Linux, and Solaris. Interstage 4.0 was released in November 2001. For more information, go to www.fujitsu.com.

Plazmic Mobile Media Platform

Finally, proof that it's a small world for Java after all. Plazmic announced that Walt Disney Internet Group Japan has launched a mobile media entertainment service on NTT DoCoMo's i-mode network using Plazmic's Mobile Media Platform. Disney customers can access a graphically enhanced entertainment service called "Magical Fortune" that runs Java applications thanks to the Plazmic Media Engine, a Java-based wireless media player. Magical Fortune is a horoscope service that allows mobile customers to enter birth-date information and be matched to one of nine popular Disney characters. Customers can subscribe to the Disney-i service to receive a daily fortune on their mobile device based on their Disney character type.

You could buy our competitor's software development tools, but then you would have to use them.



round-trip ER/UML modeling with ER/Studio®

real-time team collaboration

integrated modeling and development environment (IMDE)

Where is it written that a program has to be hard to use just because you want to speed up your software design and collaborative development efforts? The fact is, it doesn't. Embarcadero's Describe™ gives you all the benefits of the UML (Unified Modeling Language) without having to subject your team to some complex, overpriced and totally inflexible application development tool.

That should be a huge relief. After all, your tools should eliminate complexity, not add to it. Describe simplifies analysis, design and application implementation for organizations working with Java, C++ and IDL. With features that include seamless integration with popular IDEs like Sun™ Forte™ for Java,™ real-time team collaboration, and round-trip ER/UML modeling, Describe streamlines any design, development or application re-engineering effort without forcing people to change the way they like to work.

Simpler really is better. For more information, or to download a free evaluation copy, visit **www.embarcadero.com** today.

EMBARCADERO TECHNOLOGIES®

© 2001 Embarcadero, the Embarcadero Technologies logos and all other Embarcadero Technologies product or service names are trademarks of Embarcadero Technologies, Inc. All other trademarks are property of their respective owners.

Automate SOAP Calls in Java with the Proxy Pattern

Learn how to leverage the power of SOAP and the proxy pattern to develop scalable applications

When you're programming an application in which remote computers need to exchange objects over the Web—for example, an *n*-tier application in which the client sends requests to the server over HTTP—you have a choice of approaches. A prime candidate for the content of the requests is XML. However, generating and parsing the XML, both on the client and on the server, can be tedious and error prone. An alternative is to use remote method invocation (RMI). But RMI has its own drawbacks—it forces all callers to be Java compliant, and it compromises scalability by requiring a "live" remote object. A better solution for this type of situation is the Simple Object Access Protocol (SOAP). SOAP is a powerful tool that can solve communication problems between remote computers.

Henry Bequet is vice president of Internet development with Sagent. He has a M.S. in computer science and is a Sun-certified Java 2 programmer. Henry lives in Boulder, Colorado where he specializes in developing scalable distributed Web applications using XML and Java. Henry can be reached at **hbequet@sagent.com**.

SOAP is an XML-based protocol designed to exchange objects between applications over the Web. (For more information on the history of SOAP, see the sidebar "Carving SOAP.") A key advantage of SOAP is that it doesn't require a permanent connection between computers. SOAP can ride with other HTTP traffic, such as Web pages, and SOAP doesn't require the network administrator to perform any additional setup or maintenance. SOAP is a simple protocol that supports the exchange of objects but doesn't support remote invocation of objects. The big advantage of SOAP is its openness: Because it's built on XML and designed to use standard transports like HTTP and SMTP, any OS that supports those standards is available as a platform for SOAP development and deployment. For instance, ActiveX objects on a Windows 2000 machine can talk to Java objects on a Unix system via SOAP.

To help demonstrate the benefits of SOAP, we'll implement the SavonProxy class in the context of a B2B application for the fictitious company Le Savon, a manufacturer and seller of exotic soaps. The application lists the open orders of a customer and returns the status of a specific order. Each order has a unique identifier. The database of orders is implemented as a static array of Order objects. (This—the database implemented as a static array—is not what you would want in a deployed solution, but it serves its purpose for this example. For a deployed solution, the database should be in a production-worthy relational database management system (RDBMS) like ORACLE or SQL Server.) In addition to automating the generation and parsing of XML, this application provides extra benefits, like caching of client requests and local debugging of SOAP services.

The example application uses the Apache SOAP implementation. You need no prior knowledge of the Apache SOAP implementation to understand the concepts. You can download the Apache SOAP engine at http://xml.apache.org/soap.

The Code

First, let's discuss the factory. The SavonProxy implementation uses the Proxy class introduced in JDK 1.3. In a nutshell, this class provides an implementation of the proxy design pattern that works with any of your classes via the Proxy.invoke() method: You override the invoke() method to intercept all calls to your object. For a complete description of the Proxy class, please consult the JDK 1.3 documentation.

This code shows how you can use the SavonProxy.newInstance() method to create a local or remote object:

```
// use url=http://... for a proxy
// use url=inproc:// for a local object
Orders Orders = (Orders)SavonProxy.newInstance(
  url, "com.lesavon.service.
  OrderService");
```

To create a local object, specify a URL that starts with inproc://; to create a remote object, specify the URL of the SOAP service. The second argument of the call is the name of the class that implements the service (in this case, com.lesavon.service.OrderService). Note that you must pass in the fully qualified name of your class. When you specify the URL of the SOAP service, no actual object is created on the remote machine. Instead, an instance of the SavonProxy class is created. Once created, the implementation of the interface can be used in the same manner whether it represents a remote or local object.

Listing 1 shows a subset of the Order class. It's a minimal implementation of an order with an identifier and a description. The Orders.getOrderList() method returns an array of orders (see Listing 2). The Orders.runOrder() method returns an XML document. I'll get back to the serialization of these objects later. First, let's talk about the proxy implementation.

Listing 3 shows the implementation of the SavonProxy class. Aside from the factory discussed earlier, the important code is in the invoke() method. The Java 1.3 InvocationHandler interface provides the invoke() method as a way to trap all the calls going

Continued on page 86.

Listing 1 Order in the Class

```
public class Order {
// An order has an ID and a description
  private String id;
  private String description;

  // set and get methods to be used by the
  // BeanSerializer
  public void setId(String id) {
    this.id = id;
  }

  public String getId() {
    return id;
  }
}

...

public interface Orders {
  public Order[] getOrderList(String user,
    String password) throws SavonException;
  public Element runOrder(String user,
    String password, String OrderId) throws
    SavonException;
}
```

This subset of the Order class includes an identifier and a description of an order.

Carving SOAP

Microsoft (http://msdn.microsoft.com/soap), UserLand (www.soapware.org), and Developmentor (www.developmentor.com/soap) originally proposed SOAP as a protocol to handle remote calls in the BizTalk architecture. After other companies, including IBM, rejected it for being too close to the Microsoft architecture, SOAP went through some changes to open its design. It gained momentum and support from Sun, Lotus, CommerceOne, and other companies. SOAP is also used by the World Wide Web Consortium (W3C) as a starting point in the definition of the XML protocols (XP) standard. You can find more information on SOAP and XP at www.w3c.org/tr/soap. Information on HTTP is available at www.w3c.org/protocols.

Discover Your Inner Classes

Unit testing driving you nuts? Learn how using inner classes with the JUnit testing framework can bring you inner peace

by Harris W. **KIRK**

Harris W. Kirk has been developing software at pharmaceutical and financial service companies for almost 20 years and is currently completing his third project using JUnit in the role of XP coach. A Sun-certified Java programmer, Harris holds an MS degree in analytical chemistry from Purdue University. Reach him at **harris.kirk@capitalone.com**.

Unit testing can be a painful activity that few developers enjoy. This is generally true, unless the testing is automated, coded in your favorite language, and gives you new insights into the language. Enter JUnit. I have found JUnit to be a very useful tool for unit testing Java code in my projects. The ability to verify the proper operation of code through automated unit tests is becoming a valuable tool for developers. In addition, it is an absolute must while a development team is transitioning to Extreme Programming (XP). XP is a lightweight methodology that focuses on pair programming, automated testing, and rapid iterations. (For more information on XP, check out the Javatecture column "Going to Extremes," in the October 2001 issue of *Java Pro*.)

I have used JUnit on three projects and have noticed that test cases become harder to manage as the size of the test method or number of methods increases. You can use inner classes and a combination of simple techniques to more fully leverage JUnit and rapidly produce large numbers of readable test cases. This same technique can be used to specify and test for a thrown exception.

Java developers are often reluctant to perform testing when the process becomes tedious or when they cannot use Java to perform the tests. You may be surprised at how simple it is to add and run test cases in JUnit. In addition, working with JUnit can teach you some things about Java that you might not know. This is sufficient motivation for me to consider unit testing a highly beneficial practice.

You may be surprised at how simple it is to add and run test cases in JUnit

JUnit Overview

JUnit (available at www.junit.org) is a tool that allows a developer to write simple tests in Java to verify that certain conditions are met in the code being tested. This example of a test asserts that an XmlElement class produces the correct XML string:

```
XmlElement xe =
  new XmlElement("custName");
assert ( xe.toString().equals(
  "<custName></custName>");
```

JUnit operates by reflectively executing every method in a class that begins with "test" and reports the results of programmer-defined assert statements. The JUnit tool will take no action when the assertion is true. If the assertion is false, however, the tool will display the failed assertion. After using JUnit for some time, two distinct uses of the assertion mechanism within the tool have become clear to me. The first way of using assert() is to place it liberally throughout code to verify conditions before moving to the next line of code. For example, this code asserts a non-null reference prior to calling a method:

```
DBConnection conn =
  DBConnection.getInstance();
```

Install

Configure

Update

If your deployment solution doesn't do all this, you don't have a solution at all.

From client to server, Zero G's InstallAnywhere and PowerUpdate are the only software deployment solutions that you'll ever need. Our software is used by top companies like BEA, Borland, and Apple to deploy complex software to customers worldwide. **<INSTALL>** InstallAnywhere is the world's most powerful multi-platform software deployment solution. With support for multiple operating systems, 29 languages and silent, console, & GUI installations, no other solution offers broader platform support or greater flexibility. **<CONFIGURE>** Zero G's solutions come complete with everything you need to get your software running right. And, in case you need a customized solution, our products are easily extended through an open Java-based API. **<UPDATE>** PowerUpate, Zero G's complete updating solution, automatically manages the downloading and installation of the latest version of your software, eliminating the need to contact customers whenever there's a patch or new release. Plus, PowerUpdate's powerful reporting functionality delivers valuable customers usage information. **<MORE INFO>** Visit www.ZeroG.com to learn more about Zero G's complete software deployment solutions.

zero G

ZERO G

http://www.ZeroG.com/goto/ia45jp

InstallAnywhere PowerUpdate

© 2001, Zero G Software, Inc. Zero G, Zero G Software, PowerUpdate, and InstallAnywhere are trademarks or registered trademarks of Zero G Software, Inc. All other trademarks are property of their repective owners

```
assert ( conn != null );
conn.setQuery(
  "select * from customer" );
            :
            :
```

A failed assertion will cause the testing to stop. In this particular usage, that action is desirable because the assumption is that the precondition should be met before continuing.

The second use of assert() is to repeatedly invoke the same method with multiple test cases. For example, this code concatenates strings in a business-specific way:

```
String s = SomeClass.foo(
  "A", "B", "C");
assert (s.equals("A-B-C");

String s = SomeClass.foo(
  "A", "", "C");
assert (s.equals("A-C");
```

Once again, a failed assertion will cause the testing to stop. In this instance, that action is *not* desirable because the test cases following a failure will not run. It is this second use of assert() that is the main subject of this brief article.

A Growing Problem

The typical way of handling test cases in JUnit is to create a class with methods beginning with "test". For example, we could create the methods testNullAccount() and testEmptyAccount() to handle cases where the account object is null and empty, respectively. This approach is satisfactory for a few test cases, but what if the number of test cases grows to 10 or even 50? It seems that there is always another test case I should test for (guilt, guilt). Having this many methods becomes unmanageable. The situation gets ugly. It would be easy, at this point, to abandon testing entirely because of the work involved in coding.

You might work around this limitation by placing all test cases and assert statements sequentially inside a single method. This approach suffers from two limitations. The first limitation is the need to replicate the test code for every test case. Even if you have only two or three lines of code per test case, the size of the method will mushroom and become hard to maintain (and read). Listing 1 illustrates the obscurity of the test cases using this approach. The second limitation is that JUnit will stop the testing after the first failure—the remaining test cases won't be executed.

Date	Lot	Sequence	Expected Label
20010101	A13	1	A13-1 EXP. 01/01
20010101	A123	99	A123-99 EXP. 01/01
2001530	A999	""	BadDateException
20010530	A13	null	BadSequenceException

```
new TestSpec("20010101", "A13", "1", "A13-1 EXP. 01/01"),
new TestSpec("20010101", "A123", "99", "A123-99 EXP. 01/01"),
new TestSpec("2001530", "A999", "", BadDateException.class),
new TestSpec("20010530", "A13", null, BadSequenceException.class),
```

Figure 1 | Business Translation Inner classes allow us to translate business rules to object-oriented test cases.

Unit testing may become one more academic exercise that falls away under the weight of real-world pressures unless you can find a way to insert a simple structure into your test method to rapidly specify the test cases. In the remainder of this discussion, I will describe such an approach whereby you can create and run large numbers of test cases within a single JUnit method. Let's examine the desired features of test methods before moving on to the implementation:

- Readable test cases should be constructed simply (so customer can easily understand them).
- Large numbers of test cases should not require long methods.
- Large numbers of test cases should not require large numbers of methods.
- All test cases should run (the testing tool should not stop on a failed test).
- All failed test cases should be displayed in the JUnit GUI.
- The expected outcome of each test case should be encapsulated with the test case.

One of the benefits of object-oriented programming is that the computer code can be made to map more easily to the problem domain. As shown in Figure 1, the use of inner classes allows the table of business rules to be translated to code. The expected batch label of a hypothetical pharmaceutical application is shown along with the test cases. The readable test cases in this example make it easy to determine if the appropriate cases have been specified. And you can subsequently take pride in overcoming a major hurdle of testing—specifying reasonably complete test cases.

Use of Inner Classes

Initially, I found the concept quite bizarre: Why would anyone want to create a class *inside* a method? One reason has since become very clear: You may put a class inside a method if you want to create a data structure locally that can be used to populate arrays. The inner class has meaning only within the context of the method. A test case then becomes just one line in the array initializer. You then simply traverse the array to execute each test case. As you can see from Listing 2, the inner class encapsulates a test case that also includes a reference to the expected outcome. In other words, an inner class supports the easy creation of many instances of the input data and the expected outcome.

You might be thinking, "Why not declare the inner class at class scope?" Inner classes at class scope are more broadly useful because they can be subclassed and

If your back-end database isn't a good match for your front-end development, you need a new database. Caché, the high-performance database from InterSystems, is a powerful fusion of today's mainstream technologies: objects and SQL.

Unlike Oracle and other relational databases, Caché takes advantage of its efficient multidimensional data engine to implement an advanced object model. It doesn't try to hide a cumbersome relational engine beneath object-like wrappers.

Every Caché object is compatible with Java, C++, ActiveX, and other rapid development technologies. And thanks to Caché's "Unified Data Architecture," every object class is instantly accessible as tables via ODBC and JDBC.

With no mapping or middleware. That means no wasted development time. And no extra processing at run time. So not only will your applications be quick to build or adapt, they will run faster too.

Get The Right Back-End For Your Front-End.

InterSystems CACHÉ

Make Applications Faster

Download Caché for free or request it on CD at **www.InterSystems.com/match6**

© 2001 InterSystems Corporation. All rights reserved. InterSystems Caché is a registered trademark of InterSystems Corporation.

reused. Using inner classes at class scope would work just fine. However, in the spirit of XP and YAGNI (you ain't gonna need it), leaving the inner classes at method scope is the simplest thing that would work to accomplish the goals.

The number of test cases can grow rapidly when you're testing for null references, thrown exceptions, empty strings, and other unusual test cases. For example, when designing a method, it is easy to forget to ask what the method should do when it is passed a null reference. Passing null references is one of the primary ways of getting a null pointer exception. As shown in Listing 2, your test case can include a null reference in a manner consistent with the other test cases.

Another example of an unusual test case is the exception that the method may throw. The outcome of a test can be either the expected value or the class name of the thrown exception. Using the object reference in the inner class allows passing the expected exception in the array initializer. In the catch block of the test, the expected exception is compared to the actual one that was thrown and if they do not match, the test is added to the list of failed tests. The use of this approach supports—even encourages—you to write highly robust code and to test it in an elegant way.

To complete the example shown in Listing 2, it is necessary to implement one more concept. As previously mentioned, JUnit will stop upon the first failed assertion in a test method. To ensure that all test cases run, a list of failed tests is collected locally and the assertion is deferred until just before the method's exit. If the list of failed tests is not empty, the toString() method of the list elements will dutifully display each failed test case in the JUnit GUI.

Putting it all Together

When I first try to understand a class, I usually clamor to obtain an example. Having complete and meaningful test cases allows the test cases to be part of the specification. In practice, having these test cases is far more helpful in understanding how a class should be used than, say, commented code. A look at the array initializer and the executeTest() method in Listing 2 will convey the functionality of the method being tested without having to wade through its code. In addition, the test is automated so that you can retest changes to the code rapidly. The actual code I tested has been intentionally omitted from this article to illustrate the value of having the extensive test cases (it is available online at www.javapro.com). Even without the code, the intention of the code is clearly expressed through the many test cases.

So, you can see the value of using inner classes to prepare your test code when you have more than two or three test cases. The payoff arrives quickly as you add many test cases easily. Once you set up the inner class, it takes only seconds to add new test cases. You can specify and test many outcomes of a method call easily. The ability to automatically document the method's func-

Listing 1 Before Inner Classes

```
public void testCreateBatchLabelNoInnerClass() {

  try {
    BatchLabel aBatchLabel = new BatchLabel(
      "20010530", "A13", "1" );
    assert ( aBatchLabel.getBatchLabel().equals(
      "A13-1 EXP. 05/01" ) ) ;
  } catch (Exception e) {
    assert ( "Exception:" + e, false );
  }

  try {
    BatchLabel aBatchLabel = new BatchLabel(
      "20010530", "A13", "" );
    assert ( aBatchLabel.getBatchLabel().equals(
      "A13 EXP. 05/01" ) ) ;
  } catch (Exception e) {
    assert ( "Exception:" + e, false );
  }

  try {
    BatchLabel aBatchLabel = new BatchLabel(
      "20011231", "A13", "1" );
    assert ( aBatchLabel.getBatchLabel().equals(
      "A13-1 EXP. 12/01" ) ) ;
  } catch (Exception e) {
    assert ( "Exception:" + e, false );
  }

  try {
    BatchLabel aBatchLabel = new BatchLabel(
      "20010101", "A13", "1" );
    assert ( aBatchLabel.getBatchLabel().equals(
      "A13-1 EXP. 01/01" ) ) ;
  } catch (Exception e) {
    assert ( "Exception:" + e, false );
  }

  try {
    BatchLabel aBatchLabel = new BatchLabel(
      "20010101", "A123", "99" );
    assert ( aBatchLabel.getBatchLabel().equals(
      "A123-99 EXP. 01/01" ) ) ;
  } catch (Exception e) {
    assert ( "Exception:" + e, false );
  }

  try {
    BatchLabel aBatchLabel = new BatchLabel(
      "2001530", "A999", "" );
  } catch (Exception e) {
    assert( "Wrong Exception thrown: " + e,
      e.getClass().equals(
      BadDateException.class )) ;
  }

  try {
    BatchLabel aBatchLabel = new BatchLabel(
      "20010530", "913", "1" );
  } catch (Exception e) {
    assert( "Wrong Exception thrown: " + e,
      e.getClass().equals(
      BadLotException.class )) ;
  }

  // etc
}
```

As the number of test cases grows, so does the number of methods. You might think that one way to avoid this problem is to place all test cases and assert statements sequentially inside a single method. When you do this, however, the test code needs to be replicated for every test case.

CREATE THE MOST ADVANCED APPS UP TO 80% FASTER. You need to build a strategic application that will leapfrog your company over the competition. In record time. Impossible? Not for 2,000 major companies that develop apps using best-of-breed ILOG software components. ILOG's powerful Java and C++ engines can produce smarter, more intuitive applications while slashing development time 30% to 80%. This means high ROI. And more efficient, satisfied users.

DO THE IMPOSSIBLE.

ILOG™

POWERING SMARTER SOFTWARE

BUILDING SMARTER SOFTWARE FASTER.

FREE! Get our guide to building smarter apps, faster. Call 1-800-FOR-ILOG or visit www.ilog.com/jp

OPTIMIZATION
Advanced ILOG engines are the standard for dramatically improving efficiency in supply chain management, scheduling and customer self-service applications.

BUSINESS RULES
Leading-edge ILOG rule engines intelligently automate decision-making based on specific business policies, particularly for online, customer-driven applications.

INTERACTIVE VISUALIZATION
ILOG's rich, prebuilt user interface components make applications – say, managing a telecom network or workflow model – much faster to build and much more intuitive to use.

ILOG and the ILOG logotype are registered trademarks of ILOG. All other brand, product and company names are trademarks or registered trademarks of their respective owners.

WebSphere software

SUPPORTERS
OF THE NEW OPEN DEVELOPMENT
ENVIRONMENT.

ALTOWEB
ASIST
BOWSTREET
BROWSERSOFT, INC.
CAST
COLLABNET
COMMERCEQUEST
COMPUWARE
EMBARCADERO TECHNOLOGIES
HOLOSOFX, INC.
INSTANTIATIONS
INTERWOVEN
LEGACYJ
LOGICLIBRARY, INC.
MACROMEDIA
MERANT
MERCURY INTERACTIVE
MID-COMP INTERNATIONAL
MKS INC.
NEUVIS SOFTWARE
PEREGRINE SYSTEMS
QNX SOFTWARE SYSTEMS
RATIONAL SOFTWARE
RED HAT
SERENA SOFTWARE
SITRAKA
SKYVA INTERNATIONAL
STARBASE
SUSE LINUX
TELELOGIC AB
TOGETHERSOFT
VERSANT
VERSATA, INC.
WILY TECHNOLOGY

e business software IBM

To get free code and info, visit
ibm.com/websphere/ibmtools

Listing 2 After Inner Classes

```
public void testCreateBatchLabel() {

  class TestSpec  {   // start inner class
  String date, lot, sequence;
  Object outcome;

  TestSpec (String date, String lot,
    String sequence, Object outcome)  {
    this.date     = date;
    this.lot      = lot;
    this.sequence = sequence;
    this.outcome  = outcome;
  }

  public String toString() {
    return "[" + date + ":" + lot + ":" + sequence + ":" + outcome + "]";
  }

  public boolean executeTest() {
    try {
      BatchLabel aBatchLabel = new BatchLabel(date, lot, sequence);
      return aBatchLabel.getBatchLabel().equals((String) outcome );
    } catch (Exception e) {
      return ( e.getClass().equals( outcome )) ;
    }
  }

  } // end inner class

  TestSpec [] tests = {
    new TestSpec( "20010530",  "A13",  "1",  "A13-1 EXP. 05/01" ),
    new TestSpec( "20010530",  "A13",  "",   "A13 EXP. 05/01" ),
    new TestSpec( "20011231",  "A13",  "1",  "A13-1 EXP. 12/01" ),
    new TestSpec( "20010101",  "A13",  "1",  "A13-1 EXP. 01/01" ),
    new TestSpec( "20010101",  "A123", "99", "A123-99 EXP. 01/01" ),
    new TestSpec( "2001530",   "A999", "",   BadDateException.class ),
    new TestSpec( "",          "A999", "",   BadDateException.class ),
    new TestSpec( "20010132",  "A999", "",   BadDateException.class ),
    new TestSpec( "20017777",  "A999", "",   BadDateException.class ),
    new TestSpec( "20010530",  "A13",  null, BadSequenceException.class ),
    new TestSpec( "20010530",  "A13",  "W",  BadSequenceException.class ),
    new TestSpec( "20010530",  "913",  "1",  BadLotException.class ),
    new TestSpec( "20010530",  "",     null, BadLotException.class ),
    new TestSpec( "20010530",  "",     "1",  BadLotException.class ),
    new TestSpec( "20010530",  null,   "1",  BadLotException.class ),
  };

  ArrayList failedTests = new ArrayList();
  for (int i=0; i<tests.length; i++) {
    if ( ! tests[ i ].executeTest() )
      failedTests.add( tests[ i ] );
  }

  assert ("Failures: " + failedTests, failedTests.size() == 0 );
```

If you want to create a data structure locally that can be used to populate arrays, you might want to put a class inside a method. This inner class has meaning only within the context of the method. The test case in this scenario consists of one line in the array initializer. You then traverse the array to execute each test case.

tionality and test for thrown exceptions adds to the power of this approach. Finally, the example presented here will teach you more about JUnit and the Java language in the process.

For more information on and examples of the JUnit testing framework, see the Pro Shop column "Take the Work Out of Unit Testing" by Daniel F. Savarese in the October 2001 issue of *Java Pro*. JP

GO ONLINE
www.javapro.com

Use these Locator+ codes at www.javapro.com to go directly to these related resources.

JP0201 Download all of the code for this issue of *Java Pro*.

JP0201HK Download the code for this article separately.

JP0201HK_T Read this article online.

WebSphere software
IBM, WebSphere and the e-business logo are registered trademarks of International Business Machines Corporation in the United States and/or other countries. © 2001 IBM Corporation. All rights reserved.
IBM

IN THE PARALLEL UNIVERSE, DEVELOPERS' TASKS WERE COMPLEX. HERE, THE CODERNAUTS FOUND A SIMPLER WAY.

WEBSPHERE *FOR* OPEN TOOLS

SUPPORTING A TRUE OPEN ENVIRONMENT, BACKED BY INDUSTRY LEADERS.

e business software

For free code, visit **ibm.com**/websphere/ibmtools

IT'S A DIFFERENT KIND *OF* WORLD.
YOU NEED A DIFFERENT KIND *OF* SOFTWARE.

THE OBJECT OF MY DESIRE.
Adventurous object programmer seeking
DBMS who prefers the simple life, speaks my
language and can throw away the map (O-R
that is). Must be open, supportive, easy to
talk SQL to and adaptable. Please be
comfortable dealing directly with my objects.
High maintenance types with performance
issues need not reply. Let's scale the
heights together.
I think we should meet!
Cam
415-536-5394

Matisse, your dream-database, combines the power of objects and the ease of SQL to give you a high performance, extremely scalable zero administration database whose native object support saves you from the pain of object-relational mapping. But don't take our word for it...download a FREE developer's copy of Matisse at **www.fresher.com** and be eligible for a quarterly drawing to win a Compaq iPaq™.

Fresher™ Matisse®
The Object Developer's Database

no O-R mapping • native object support • ANSI SQL support • zero administration • 24x365 proven reliability • addictively easy-to-use • small footprint • XML ready

©2001 Fresher Information Corp. All rights reserved. Fresher and Matisse are trademarks or registered trademarks of Fresher Information Corp. Compaq and iPaq are trademarks of Compaq Computer Corporation.

Sales: 415-536-5394 www.fresher.com

Java: Closer to .Net than You Think

As .Net unfolds, similarities to Java are striking. Coincidence?

by James W. COOPER

James W. Cooper is the author of 14 books. *Java Design Patterns: A Tutorial*, was published by Addison-Wesley in January 2000 and his new book *Visual Basic Design Patterns: VB 6.0 and VB.NET* was published in November 2001.

Whether or not you're a heavy user of Microsoft's .Net strategy, you may know that the design of Java influenced the language structure of .Net in significant ways. In fact, I have discovered that I can write almost exactly the same code in both systems. .Net is still unfolding, and the part that has been completely explained comprises far more than language design. However, by understanding this language design, you'll gain some valuable insights into where .Net can go.

The two most important languages in the .Net system are C# and VB.Net, the later implementation of Microsoft's wildly successful Visual Basic language. Although you may think you can dismiss VB, you need to realize that it is probably the most widely used computer language in the world (and this includes all platforms). The unique point about the .Net system is that C# and this latest version of VB are just two sides of the same coin. They use the same basic system libraries and compile to the same intermediate code. This intermediate code is then interpreted at execution time much like how the Java bytecodes are interpreted by the Java virtual machine. In fact, all of the .Net languages compile to the same intermediate language, called MSIL, and are then executed by the Common Language Runtime (CLR). Within this system, both C# and VB.Net now provide garbage collection much like Java does. Although apparently this intermediate language code can be compiled at execution time into binary Intel code, this is not the current default model.

It seems to me that this approach can be valuable to Microsoft in moving to 64-bit Intel computing. It does not need to provide specially compiled versions of code written for .Net. It needs to provide only a run-time system that is optimized for that platform, and all .Net code will run unchanged. Of course, this also provides a convenient pathway to different platforms such as Linux and Macintosh, should Microsoft wish to pursue competing in that environment.

The greatest strength (and weakness) of C# and VB.Net is the easy-to-use visual builder environment that allows you to make nice-looking Windows applications in just a few clicks and drags. All the code to support these windows components is generated automatically for you, and the entire layout is handled under the covers. Even today, no Java development environment does this as well, or is as well integrated with the Windowing system. Of course, the weakness of this system remains that it is Windows-only, and this fact can be a drawback for cross-platform development projects.

> It is not entirely unreasonable to convert a Java program to VB.Net by pasting the code into the UI designer and editing it

The code that generates the windows is now within your classes, and you can change it or generate windows using the same kind of coding that the designer GUI produces. VB and C# do not have layout managers at this time, so all of the layouts are in absolute terms. The controls support a kind of layout using anchors and the like, but it is not brought to the surface so you can use it in the IDE very conveniently. We can, of course, argue endlessly as to which way is better, but within the Windows environment, fixed layout is generally accepted as adequate.

What is C# Like?

The C# language has the same overall syntax as C, C++, and Java: It uses braces and has the exact same syntax for all the fundamental language elements. C# is considerably more like Java than C++, in that it has classes with methods and constructors, and it specifically allows only single inheritance. Like Java, C# also supports inter-

> The unique point about the .Net system is that C# and this latest version of VB are just two sides of the same coin—they use the same basic system libraries and compile to the same intermediate code

faces. In most C# programming, you don't use pointers at all, although you can create *unsafe* code blocks where pointers are allowed. Probably most reassuringly, the code you write in C# is so much like Java that you can almost paste your Java code into the Visual Studio.Net development environment and compile it. The only difference is in the capitalization of some of the methods. Here is some C# code for splitting a string at the first space:

```
private void splitName(
  string name) {
  int i = name.IndexOf (" ");
  if (i > 0 ) {
    frname = name.Substring (
      0, i).Trim();
    lname = name.Substring (
      i + 1).Trim();
  }
}
```

For comparison, you would write this same code in Java as:

```
private void splitName(
  String name) {
  int i = name.indexOf (" ");
  if (i > 0) {
    first = name.substring(
      0, i).trim();
    last = name.substring(
      i + 1).trim();
  }
}
```

In general, the method names are much the same in the common system level classes where you end up writing most of your code. The GUI and file I/O classes are rather different, but because the GUI code is generated mostly for you, and because you tend to encapsulate file code in convenient ways, the differences are small indeed.

Inheritance in C#

C# has the same single inheritance structure as Java, although the syntax is more turgid and C++-like. For example, to create a class derived from SwimData, you write:

```
public class SexSwimData:SwimData
```

and to pass data to the base class's constructor, you write a derived class constructor that refers to the base class using reserved *base* keyword.

```
public SexSwimData(string
  filename):base(filename){}
```

I think that the Java approach is easier to read:

```
public class SexSwimData
  extends SwimData {
  public SexSwimData(
    String filename) {
    super(filename);
  }
```

C# also allows you to create interfaces rather like Java:

```
public interface MultiChoice {
  ArrayList getSelected();
  void clear();
  Panel getWindow();
}
```

However, the C# syntax does not make it easy to distinguish inheritance from implementation—it looks just the same. Here we create a class called ListChoice that *implements* the MultiChoice interface shown in the previous code snippet:

```
public class
  ListChoice:MultiChoice {
```

I think this could be confusing.

Overriding Base Class Methods

C# follows the C++ custom of requiring you to declare that a method can be overridden in derived classes by declaring the base method as *virtual*. Here we make a draw method overridable:

```
public virtual void draw(
  Graphics g) {
  g.DrawRectangle (rpen, x,
    y, w, h);
}
```

Then, in the derived class, you must specify that you are overriding that method using the *override* keyword:

```
public override void draw(
  Graphics g) {
  base.draw (g);
  //draw one rectangle
  g.DrawRectangle (rdPen,
    x +5, y+5, w, h);
}
```

As you can see, you can also call the base method using the *base* keyword.

If you don't want to or can't make the base class method virtual, you can use the *new* keyword to create a method that replaces all methods of that name and any signature in the base class with a new one:

```
public new void draw(Graphics g) {
  g.DrawRectangle (rpen, x,
    y, w, h);
  g.DrawRectangle (rdPen, x +5,
    y+5, w, h);
  }
```

In this case, you no longer can call the parent method in the base class.

VB.Net works in exactly the same way, except that the keywords are much more awkward:

```
Public Overridable Sub draw(
  ByVal g As Graphics)
  g.DrawRectangle(rpen, x,
    y, w, h)
End Sub
```

and

```
Public Overrides Sub draw(
  ByVal g As Graphics)
```

```
  MyBase.draw(g)
  g.DrawRectangle(redPen, x + 4,
    y + 4, w, h)
End Sub
```

To replace a base method instead of overriding it, you use the *Shadows* keyword in VB.Net instead of the *new* keyword used in C#.

Exceptions

C# provides a wide array of standard exceptions that the system may throw when errors occur. These include NullReferenceException, DivideByZeroException, FileNotFoundException, and so forth. The try–catch syntax is analogous to that in Java:

```
try {
  ts = new StreamReader
    (fileName);
  opened=true;
  }
  catch(FileNotFoundException e )
  {
    Console.WriteLine (
      e.Message );
  }
```

One thing that I find a major disadvantage is that C# (and VB.Net) do not have a *throws* keyword. There is no way to indicate that a method can throw an exception and that you must prepare your code for that eventuality. If some class you call throws an exception, you find out when it happens, or by reading the code. So there is no way for the compiler to enforce the need to catch exceptions.

Comparing VB.Net

The VB.Net system is supported by the Visual Studio.Net development environment. You can choose to build either C# or VB programs (or C++). The same GUI designer is used for both, but different language-specific code is generated. This is the first version of VB to support inheritance, although VB has supported interfaces awkwardly in the past two versions. Because the libraries with which this new version of VB interacts are the same ones that C# uses, you now have a bunch of new, preferred class methods to use in VB.

For example, to split a string in VB, we might write:

```
Private Sub splitName(
  nm As String)
  Dim i As Integer =
    nm.IndexOf(" ")
    If i > 0 Then
      Frname = nm.Substring(
        0, i).Trim()
      Lname = nm.Substring(
        i + 1).Trim()
End If
```

Not only is this much the same as in C#, it is still much the same as in Java. It is not entirely unreasonable to convert a Java program to VB.Net by pasting the code into the UI designer and editing it to be correct. In fact, I have done this, converting all of my Java design pattern examples to VB.Net in a very short time. Converting them to C# is even easier.

Inheritance in VB.Net is syntactically clunky, but the development environment manages a lot of it for you. You can create a derived class using the Inherits keyword:

```
Public Class Stocks
  Inherits Equities
```

and the Stocks class will inherit all the methods of the Equities class. Likewise, you can define an interface:

```
Public Interface MultiChoice
Function getSelected() As
ArrayList
  Sub clear()'
    clear all selected
  Function getWindow() As Panel
End Interface
```

and specify classes that implement that interface more clearly than you can in C#:

```
Public Class ListChoice
  Implements MultiChoice
```

However, VB forces you to add some awkward syntax to every method that implements this interface:

```
Public Sub clear() Implements
  MultiChoice.clear
  lst.Items.clear()
End Sub
```

but the development environment will generate this boilerplate for you.

VB.Net, of course, also supports the same sort of file manipulations and exceptions as C#, because it is effectively the same language. This is great for C# developers, but does represent a sea change for earlier generations of VB programmers who now have to learn a great number of new method names and some changes in syntax.

> The .Net languages bear a strong resemblance to Java, and one might argue that Microsoft's strategy is to use this similarity to tempt Java programmers to move to its system

As you can see, the .Net languages bear a strong resemblance to Java, and one might argue that Microsoft's strategy is to use this similarity to tempt Java programmers to move to its system. For certain classes of Windows applications, this might make sense. However, the contrary is also true. Java is now a successful, widely used, multiplatform language, and anyone who learns the .Net languages can now move out of the fold as easily as they moved in. The .Net system may also turn out to be a great training ground for moving VB-style programmers to Java! JP

Use this Locator+ code at www.javapro.com to go directly to this related resource.

JP0201JT_T Read this article online.

Creating an Infrastructure for Testing Servlets

by Adam Kolawa

Testing a servlet via an HTTP connection (client-side testing) is an important part of servlet testing. However, I have seen some groups delay this testing until a fair amount of development had been completed because they thought that the servlets first needed to be deployed to a common server. This is an inefficient practice that hampers productivity. If you delay your testing until later in the development process, you might reuse problematic code and build on it before you realize that you have a problem. If so, a simple one-minute, one-line change can blossom into an extensive debugging and/or rewriting nightmare.

If you establish an appropriate infrastructure, you can perform client-side testing on a servlet immediately after you compile it and perform unit testing. First, make sure each developer has access to an individual Web server running on his or her machine. Next, configure your build process so that you can run your servlets immediately after compilation, with no extra steps. To do this, implement an efficient way to move `.class` files as soon as they have been compiled. One way to streamline this process is to configure your build process to place newly built class files into the directories from which your servlet engine runs them; another way is to configure your servlet engine to look into your development directory. You can further speed up the testing process by configuring your servlet engine to recognize updates and execute the current code automatically.

If an application has dependencies on other components (for instance, databases), arrange your environment appropriately by sharing a common database, having personal databases, or stubbing out database access.

Naturally, testing early doesn't eliminate the need for later application-wide testing. However, it does improve the overall efficiency of your development process and make the later phases of testing decidedly less painful.

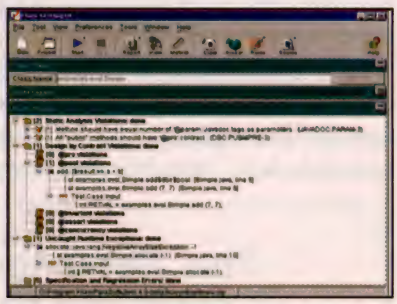
Adam Kolawa, Ph.D., is Chairman and CEO of ParaSoft. You can reach him at ak@parasoft.com

ParaSoft®
jtest®

Click once for automatic Java™ unit testing!

"It is fundamental that every piece of Java code is tested at the class level as soon as it's been written, but writing test cases is tedious and time consuming. Jtest lets me do this automatically and gives me the confidence that my code has been thoroughly tested. I love it!"
—Mansour Raad, DiscoverCast, Inc.

The development community endorses unit testing, coding standards, and Design by Contract™ because they increase software reliability. Until now, these practices were too costly and labor-intensive to implement.



Jtest is user-friendly; you can start finding bugs automatically with the click of a button.

Jtest® eliminates these obstacles, giving even the busiest developer the power to automate these critical procedures, including—for the first time in software development history—fully automated black-box testing. Jtest is an automatic unit testing tool for Java. With the click of a button, Jtest automatically performs the critical testing procedures discussed below, slashing debugging time and taking testing to the next level.

Black-box (functionality) testing ensures that methods in your class function properly. Jtest fully automates this by reading the specification information built into the class with the Design by Contact (DbC) language, then automatically creating and executing test cases that verify the functionality described in the specification. You can also add test cases by adding method inputs for simple test cases, or creating Test Classes for complex ones.

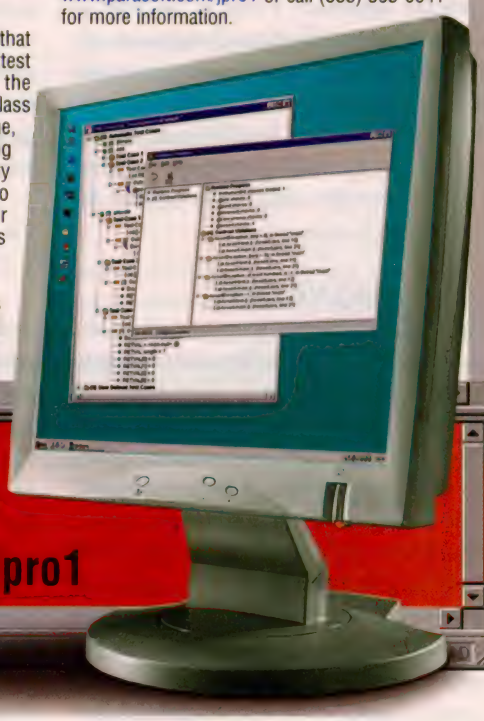
White-box (construction) testing produces robust classes that can withstand punishment. Jtest automates this by generating and executing test inputs based on a class' structure, ensuring that unexpected inputs do not cause crashes. When an input generates an uncaught runtime exception, Jtest reports the arguments, stack trace, and calling sequence, pinpointing an error's exact location.

Regression testing verifies that code performs according to specification after changes are made. Jtest performs automatic regression testing at the class level, monitoring your existing code's integrity earlier than ever thought possible. To do this, simply restore a test and click a button or integrate batch-mode Jtest into your nightly builds to ensure that errors are found quickly.

Jtest prevents errors by automatically checking whether your code adheres to over 240 industry-respected coding standards. You can extend and customize Jtest's coding standards with the RuleWizard™ feature, which lets you graphically create custom standards.

Jcontract™ is a new tool that checks DbC contracts at runtime. It is complementary to Jtest and can be used to verify that code works correctly at the system level.

To use Jtest and Jcontract for your Java projects, download a demo today at www.parasoft.com/jpro1 or call (888) 305-0041 for more information.



ParaSoft®

Download your FREE DEMO at: www.parasoft.com/jpro1

© 2002 ParaSoft Corporation, Monrovia, CA, USA. ParaSoft and Jtest are registered trademarks of ParaSoft Corporation. RuleWizard and Jcontract are trademarks of ParaSoft Corporation. All other brands, services and companies are trademarks, registered trademarks or servicemarks of their respective holders in the U.S. and/or other countries.

The Lowdown on Uploads

Expand the usefulness of your Web site: Add file upload capabilities with this generic Java component

by Alexandre CALSAVARA

Alexandre Calsavara is an independent contractor with extensive experience in embedded systems, C/C++, and Java. He also maintains a Web site about computer programming at www.cpworld.cjb.net. He has a bachelor's degree in systems analysis and can be reached at **apcal@cpworld.every1.net**.

The capability of uploading files using HTML forms has been around for some time. It is part of the HTML standard, is supported by all major browsers, and has many uses among Web applications.

The great advantage of file upload is that you can send large amounts of data beyond what you could using just regular forms. File upload can make many tasks—archiving files, submitting resumes for job applications, maintaining your Web site, and submitting data for processing, for example—easier. Oddly enough, the major solutions for server-side programming, including Java, don't provide any tool or built-in feature to handle file uploading.

Because HTML file upload employs the same format as multipart messages as defined by the Multipurpose Internet Mail Extensions (MIME) standard, you could use JavaMail to process such requests (that was the solution used in some projects I worked on). However, using JavaMail is not intuitive, and it adds overhead to your program.

Therefore I developed a reusable, generic Java component specifically designed to handle file uploads. The component supports uploading any number of files of any size, and it allows you to monitor the progress of the upload. It is equally well suited to be used both with Java servlets and JavaServer Pages (JSP).

HTML File Upload

The HTML standard defines an element, FORM, specifically designed to enter and send data, including files, to a server for processing. (For more information on the HTML 4.01 forms specification, see Resources.) To create a form capable of uploading files you must follow these steps:

- Set the value of the "method" attribute of the FORM element to "post".
- Add the attribute "enctype" with the value "multipart/form-data" to the FORM element.
- Add one or more INPUT elements with the type "file" to the form.

When the browser finds an INPUT element with the type "file", it creates a file select control, which is usually rendered as an input box with a button that allows you either to select or type the name of the file to send. Of course, any number of regular controls can be present, as well. A simple form (ServletExample.html) allows you to upload a file (see Listing 1). Figure 1 shows how the form is rendered by a browser.

RESOURCES

HTML 4.01 Specification, Chapter 17, "Forms"
www.w3.org/TR/1999/REC-html401-19991224/html40.txt

RFC 2046, Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, Section 5.1, "Multipart Media Type"
www.ietf.org/rfc/rfc2046.txt

RFC 2045, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*
www.ietf.org/rfc/rfc2045.txt

RFC 822, *Standard for the Format of ARPA Internet Text Messages*
www.ietf.org/rfc/rfc822.txt

RFC 2388, *Returning Values from Forms: multipart/form-data*
www.ietf.org/rfc/rfc2388.txt

Java Servlet Specification, version 2.3
www.jcp.org/aboutJava/communityprocess/final/jsr053

RFC 2616, *Hypertext Transfer Protocol — HTTP/1.1*
www.ietf.org/rfc/rfc2616.txt

Listing 1 File Upload

```
<HTML>
<BODY>
<FORM action="servlet/DisplayParameters"
    enctype="multipart/form-data"
    method="post">
  <P>
  What is your name?
    <INPUT type="text" name="submit-name"><BR>
  What file are you sending?
    <INPUT type="file" name="submit-file"><BR>
  <INPUT type="submit" value="Send">
    <INPUT type="reset">
</FORM>
</BODY>
</HTML>
```

This simple example shows the main components of a form that can be used to upload a file. Notice the attributes "enctype" and "method" of the FORM element and the INPUT element with type "file".

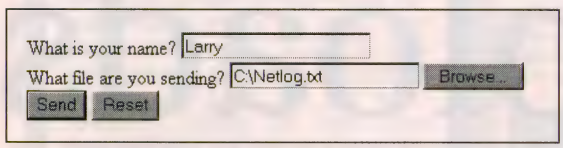


Figure 1 | File Select Controls This is how file select controls are displayed by Internet Explorer 5. Other browsers may display them differently.

Forms using the multipart/form-data encoding type are submitted following the general guidelines defined for all multipart media types (see Resources for more information on these MIME guidelines). The multipart media type is just a series of MIME entities (parts) delimited by a boundary (see Figure 2). The boundary is a unique line of text that is guaranteed not to occur inside any part, so that it can safely be used to mark the limits of each part.

The syntax of each part is the same as that of a MIME message (as defined by RFC 2045 and RFC 822; see Resources), but different in meaning. Each part is composed of a header area, a blank line, and a body area (see Figure 3). The header area contains one or more headers and is delimited by the blank line. The body area is everything following the blank line up to the boundary.

The header area contains several headers, which are text lines that define the content and format of the body area. Each header follows the same general format and has a header name, a colon, a header value, and, optionally, several parameters. Each parameter has an attribute, an equal sign, and a value, possibly quoted, and is preceded by a semicolon. Figure 4 shows the format of a header.

Subtypes of the multipart media type define additional semantics and restrictions for each application. The subtype form-data is designed to send data from forms for processing (as defined in RFC 2388; see Resources). Although it can be used for any form, it is particularly well suited to send data from forms that contain file select controls; that is, it is well suited to uploading files.

According to the form-data subtype, each field of the form is sent as a separate part. The header area of each part must contain the header Content-Disposition, with the value form-data. Additionally, the Content-Disposition header must contain the attribute "name" with the name of the field, as specified in the form, as its value. If the field is a file, the attribute "filename" must be present as well, with the file name entered in the control as its value. The header Content-Type may also identify the file contents—each type of file has its own value. The headers and parameters used by the form-data subtype are summarized in Table 1 and Table 2.

Figure 5 shows an example of what a browser could send if you enter the data displayed in Figure 1 into the form presented in Listing 1. The boundary is AaB03x, and it delimits the start of each part and the end of the last part (and of the data). The first part corresponds to the text field submit-name, as indicated by the "name" attribute of the Content-Disposition header. The content of the field—Larry—is sent as the body area of the part. The second part corresponds to the file select control submit-file, as you can see from the "name" attribute. Moreover, you know that it corresponds to a file because the Content-Disposition header presents the attribute "filename", which is the original file name, C:\Netlog.txt, as entered into the control. The file contents are sent as the body area of the part. Additionally, the Content-Type header tells you that the file contains text/plain data.

These explanations are simplifications of the complete standards, but they describe enough for practical purposes and help you understand my implementation. If you want to learn more, the Resources section contains references to the original documents.

The Servlet Model

Before examining my solution for the file upload problem, it is useful to recap how the servlet model works to understand the traps, pitfalls, and tradeoffs involved, and the de-

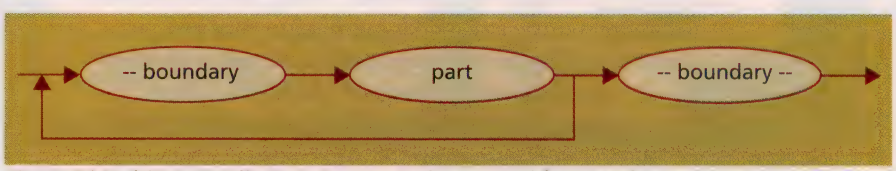


Figure 2 | Multipart Media Type Syntax In this general format of the multipart media type, the boundary is preceded by two dashes and delimits each part. The last boundary is also followed by two dashes to mark the last part.

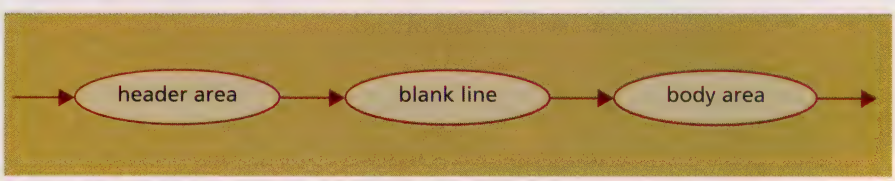


Figure 3 | Part Syntax Each part of the multipart media type contains a header area, a blank line, and a body area. The blank line acts as a delimiter between the header and the body.

The ***Fastest and Most Scalable***
Java Messaging Server

FioranoMQ 5

now adds:

XML Content-based Routing to deliver personalized content in real-time to tens of thousands of concurrent subscribers using XPath Selectors.

A unique and proven high-performance ***File-based Datastore*** together with enterprise ready features including ***LDAP and REALMS integration***, ***SOAP support***, pluggable ***Scalable Connection Management (SCM)*** module and more, makes FioranoMQ the messaging infrastructure of choice for the world's leading financial and telecommunications companies including AT&T Wireless, Morgan Stanley, Motorola and others.

Fiorano

Distributed Computing Infrastructure

Download your ***Free Developer License***
http://www.fiorano.com

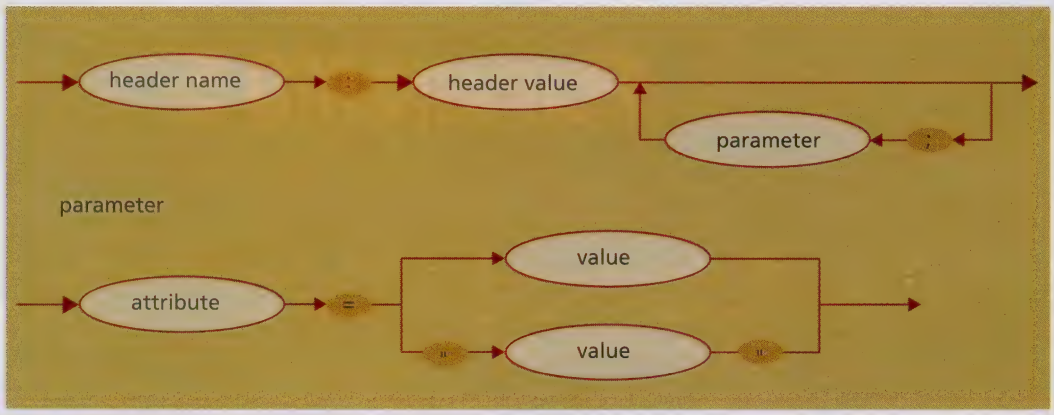


Figure 4 | Header Syntax All headers in the header area follow the same general format. The header name defines its meaning and valid parameters.

sign decisions I made. Keep in mind that I am focusing the servlet model as applied to the processing of HTTP requests, represented by the class HttpServlet (see Resources for the servlet specification).

An HTTP transaction has a request and a response. The browser sends the request to the Web server as the result of several actions, such as the user navigating to a page, clicking on a link, or submitting a form. The server receives and processes the request and sends back a response. Both the request and the response are similar in format to a MIME entity (we saw the MIME format in Figure 3). Both have a header area, a blank line, and a body area. The header area and headers follow the same format also (as we saw in Figure 4). Of course, the HTTP protocol defines a set of headers specific for HTTP transactions, both for requests and for responses (RFC 2616; see Resources).

The class HttpServlet does the job of processing HTTP requests for you. It reads, parses, and encapsulates the HTTP request in an object that implements the interface HttpServletRequest. The interface HttpServletRequest and its superinterface, ServletRequest, provide several methods to access the headers of the request's header area and the contents of its body area (see Table 3). Notice that the methods that return the value of a header also return any parameters. The task of parsing and extracting the parameters is left to the servlet.

When a form is submitted, the browser sends an HTTP request to the Web server with the fields of the form. If the "method" attribute of the FORM element is get(), the fields are sent encoded as part of the request's URL; if the method is post(), the fields are sent as the body of the request. In the latter case, the format of the data depends on the enctype attribute.

The HTML standard defines two values for the enctype attribute: multipart/form-data and application/x-www-form-urlencoded. The multipart/form-data encoding type is used with forms that have file select controls to upload files.

The application/x-www-form-urlencoded encoding type is the most common, used when the form does not have any file select controls. This encoding type is also used when the enctype attribute is omitted or when the method attribute is get().

The Content-Type header of the HTTP request identifies the type of the request's body area or, in other words, the encoding type used to submit a form. If the method attribute is get(), the Content-Type header is omitted because the request's body is empty since the fields are sent as part of the request's URL. If the method is post(), the Content-Type is the value of the enctype attribute; that is, application/x-www-form-urlencoded or multipart/form-data.

When the class HttpServlet receives a request, it parses and extracts any fields from both the request's URL and from its body, if the request's Content-Type header is application/x-www-form-urlencoded. The extracted fields can be accessed through an API defined by the ServletRequest interface (see Table 4). According to the servlet specification terminology, in this context the fields are called parameters. Figure 6 illustrates the process.

If Content-Type is multipart/form-data, the HttpServlet class doesn't process

Header	Value	Description
Content-Disposition	form-data	Must be present for each part. Identifies the part as containing data from a form field.
Content-Type	MIME type	May be present in parts corresponding to file select controls in the form. In this case, it identifies the type of the file, according to MIME standards.

Table 1 | Subtype Headers The form-data subtype specifies some headers in order to define the contents of each part. Here we see these headers, their use, and meaning.

Parameter	Value	Description
name	field name	This attribute is always present. It contains the name of the form field corresponding to that part, as specified in the form.
filename	file name	This attribute is used only for parts corresponding to file select controls in the form. In this case, it contains the name of the file as entered in the control.

Table 2 | Header Parameters Parameters or attributes are used to specify additional information for a header. The form-data subtype defines some parameters to provide further information for the Content-Disposition header.

The Hottest Magazine for the Hottest Developers in the Enterprise

If you're a professional developer programming in the hottest language in the fastest growing development market in the universe, then you need hands-on, code intensive, product-oriented information that comes only from *Java Pro* magazine. Harness Java's unique capabilities in YOUR enterprise with no-holds-barred editorial content.

Don't miss this special offer! Get 13 information-packed issues–Call Today!

Java™ Pro: get beyond the hype

- Practical techniques to amplify Java performance
- Independent, objective product reviews to help you find only the best products for your needs
- Complete source code examples using JavaBeans, servlets and more
- Tips and tricks for Java power programmers

Java Pro: is the professional developers independent source for building serious Java applications

Order online today
www.javapro.com

Subscribe by phone
1-866-387-5776

FTP FAWCETTE TECHNICAL PUBLICATIONS
209 Hamilton Ave.
Palo Alto, CA 94301

One year is thirteen issues. Annual cover price is $77.35. Subscription begins upon receipt of payment. Please allow four-to-six weeks for delivery of first issue. International subscriptions must be prepaid in U.S. dollars plus postage. Canada/Mexico, add $18/year; other countries add $44/year for airmail. Canadian GST included. *Java Pro* is published by Fawcette Technical Publications, Inc. Java is a trademark of Sun Microsystems, Inc.

the request's body. The task of parsing and extracting the fields and files sent by the form is left to the servlet. Notice that if Content-Type is multipart/form-data, the boundary used to delimit each part is sent as a parameter of the header. So, taking the example shown in Figure 5, the Content-Type header of the request would be:

```
Content-Type: multipart/
   form-data; boundary=AaB03x
```

MultipartRequest Component

Because the HttpServlet class doesn't parse multipart/form-data requests, I created a generic, reusable component to facilitate the processing of such requests so that I wouldn't have to code the same logic over and over again. The result is the class MultipartRequest contained in the file MultipartRequest.java (available online at www.javapro.com).

The class is modeled as a JavaBean, so it is easy to use both in a servlet and in a JSP page. Basically, the MultipartRequest class takes an HttpServletRequest object, parses it and makes any fields available through an API, much like the ServletRequest interface. Table 5 lists the MultipartRequest class's complete API.

Because a String is not a suitable representation for files, the MultipartRequest class encapsulates any file values as MultipartRequest.File objects, which provide an API to access the file contents and other useful properties (see Table 6). That means that values of fields are represented either as a String or as a MultipartRequest.File object, rather than the uniform String representation employed by the ServletRequest interface.

For this reason, the methods getParameter(String) and getParameterValues(String)

```
--AaB03x
Content-Disposition: form-data; name="submit-name"

Larry
--AaB03x
Content-Disposition: form-data; name="submit-file"; filename="C:\Netlog.txt"
Content-Type: text/plain

NETDI: ClassInstall (0x6 on 0x1a16:0x58) on  at
NETDI: dif_FirstTimeSetup
NETDI: SetDefaultTransport: Couldn't determine transport
NETDI: ClassInstall(0x6) end
NETDI: NETDI: lpRegLogConf = 0x0
NETDI: ClassInstall (0x9 on 0x1a16:0x58) on  at
NETDI: ClassInstall(0x9) end
NETDI: ClassInstall (0xa on 0x1a16:0x58) on  at
NETDI: ChangeLine: No matches found.
NETDI: ClassInstall(0xa) end
NETDI: ClassInstall (0xc on 0x1a16:0x58) on  at
NETDI: ClassInstall(0xc) end
--AaB03x--
```

Figure 5 | Multipart/Form-Data Media Type In this example of the multipart/form-data media type, notice the correspondence between the parts and the fields of the form—how the boundary (AaB03x) delimits the parts and how the headers are used to describe each part.

Method	Description
getContentLength()	Returns the value of the Content-Length header, or -1 if the Content-Length header is not present in the request.
getContentType()	Returns the value of the Content-Type header, or null if the Content-Type header is not present in the request.
getDateHeader(String)	Returns the value of a header field, which must be a date, or -1 if the header is not present in the request. The date is parsed and returned as a long value.
getHeader(String)	Returns the value of a header field as a String. If the header is not present in the request, it returns null.
getHeaderNames()	Returns an enumeration of all header field names present in the request.
getInputStream()	Returns a binary stream, as a ServletInputStream object, that encapsulates the request's body area and can be used to retrieve its contents.
getIntHeader(String)	Returns the value of a header field, which must be an integer, or -1 if the header is not present in the request. The integer is parsed and returned as an int.
getReader()	Returns a character stream, as a BufferedReader object, that encapsulates the request's body area and can be used to retrieve its contents.

Table 3 | Head and Body Both the header and the body area of an HTTP request are available through a specific API implemented by the object encapsulating the request passed to the servlet.

Method	Description
getParameter(String)	Returns the value of a field as a String, or null if the field does not exist. You should use this method only when you are sure the field has only one value. If the field might have more than one value, use getParameterValues(String).
getParameterNames()	Returns an Enumeration of String objects containing the names of the fields contained in this request.
getParameterValues(String)	Returns an array of String objects containing all of the values the given field has, or null if the field does not exist.

Table 4 | Form Fields The fields of forms submitted using the get() method or the application/x-www-form-urlencoded encoding type with the post() method are available through an API defined by the ServletRequest interface, as parameters of the request.

return field values as the generic Object class and must be cast to the appropriate type. Convenient methods getStringParameter(String) and getFileParameter(String) allow you to access a value as a String or as a MultipartRequest.File object if you are sure about its type.

The most important method is setRequest(HttpServletRequest). It saves a reference to the HttpServletRequest object encapsulating the request that can be retrieved later by the method getRequest(). This way the MultipartRequest class allows you to use the functionality already provided by HttpServletRequest without duplicating its API.

Besides that, the method setRequest(HttpServletRequest) reads and parses the request, saving any fields so that you can access them later using the MultipartRequest API. This is done by the method parseRequest().

First, parseRequest() parses the Content-Type header of the request and extracts its boundary attribute. If the Content-Type header or the boundary attribute is not found, the request is not multipart/form-data and the method does nothing. Otherwise, it calls the method read(PushbackInputStream,String), which reads and returns one character of the request, detecting boundaries at the same time, until it finds the first boundary, or the beginning of the first part.

Then, parseRequest() parses the header area of the part, with the aid of readLine-

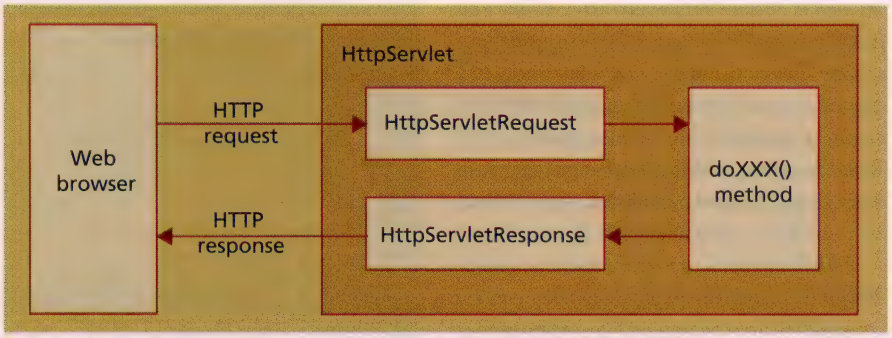


Figure 6 | Servlet Model The HttpServlet class performs the low-level processing of HTTP transactions. It receives an HTTP request and encapsulates it through an HttpServletRequest object, which is passed to the doXXX() method. The response is generated from a HttpServletResponse object returned from the same method.

Method	Description
get(int)	Returns the MultipartRequest object with the given ID, or null if it doesn't exist or was released.
getExpiration()	Returns the expiration of a MultipartRequest object as a long value. The time unit is the same used by the method System.currentTimeMillis().
getFile()	Returns the name of the file being uploaded, or an empty string if no file is being uploaded.
getFileParameter(String)	Convenient method that returns the value of a file field. Returns the value of the given field as a MultipartRequest.File, or null if the field does not exist. If the field has multiple values, returns just the first one. Use the method getParameterValues(String) to get all values.
getID()	Returns the ID of a MultipartRequest. Each object is guaranteed to have a unique ID.
getParameter(String)	Returns the value of a given field as an Object, or null if the field doesn't exist. The value of a field is a String or a Multipart.File object. If the field has multiple values, it returns just the first one. Use the method getParameterValues(String) to get all values.
getParameterNames()	Returns an Iterator that iterates over the names of the fields contained in a MultipartRequest object. The names of the fields are String objects.
getParameterValues(String)	Returns an array of objects containing all of the values the given field has, or null if the field does not exist. The values of a field are String or Multipart.File objects.
getProcessed()	Gets the number of bytes of the request's body already processed. This method can be called by another thread.
getRequest()	Gets the HttpServletRequest corresponding to a MultipartRequest object.
getStringParameter(String)	Convenient method that returns the value of a "normal" field (one that is not a file). Returns the value of the given field as a String, or null if the parameter does not exist. If the field has multiple values, it returns just the first one. Use the method getParameterValues(String) to get all values.
getTotal()	Returns the total number of bytes of the body area of the request. This method can be called by another thread.
release()	Releases a MultipartRequest object and all of its parameters. This method should be called when the object is not needed anymore. If an object is not explicitly released before its expiration, it will be automatically released when it expires.
setExpiration(long)	Sets the expiration date for a MultipartRequest object. The expiration date is specified using the same base time as the one used by the method System.currentTimeMillis().
setRequest(HttpServletRequest)	Sets a request for a MultipartRequest object. Parses the request and populates the object with the fields from it. Notice that MultipartRequest objects are meant to be used only once; that is, they can process just one request.

Table 5 | MultipartRequest API The parameters of the MultipartRequest API are similar to those of the ServletRequest interface. Additional methods are provided to monitor the status of the upload.

(PushbackInputStream) and parseHeader(), looking for the Content-Disposition header. When found, it extracts the "name" attribute of the header, which is the name of the field, and tests if the "filename" attribute exists.

If it exists, the part is a file, so parseRequest() creates an instance of the MultipartRequest.File class with the value of the "filename" attribute as the file name. Then it saves the MultipartRequest.File object just created as the value of the field and gets an output stream, using the method getOutputStream() of the MultipartRequest.File class, to save the contents of the part. If the "filename" attribute is not present, parseRequest() just creates a ByteArrayOutputStream to save the contents of the part.

If parseRequest() finds a Content-Type header while processing the header area, the value of the Content-Type header is just saved. Later, when the part is completely processed and MultipartRequest finds that the part corresponds to a file (because the Content-Disposition header is also present), then the previously saved value of the Content-Type header is set as the file type in the MultipartRequest.File object created. This process is necessary because the headers have no guaranteed order, so Content-Type may appear before the Content-Disposition header. In this case, whether or not the part corresponds to a file is still unknown. In addition, the MultipartRequest.File object has not yet been created.

Notice that the parsing of headers is performed by the method parseHeader(). This method takes a header, extracts the header name, value, and any parameters, and returns them as a Map object where the keys are the header name and any attributes, and the values are the corresponding values.

After processing the header area, parseRequest() reads the contents of the part and writes it to the output stream created according to the type of the part. When the boundary (the end of the part) is detected by the method read(), the output stream is closed. If the part isn't a file then it is a "normal" field, and the output stream created saves the data to a byte array. So parseRequest() gets the data as a String and saves it as the value of the field. If the boundary detected is the last one, there are no more parts, and parseRequest() returns; otherwise, the process starts again.

While processing the request, parseRequest() updates properties that can be queried by another thread (another servlet or JSP) to monitor the status of the processing. This functionality is particularly useful when uploading big files. The properties that can be queried are the total size of the request (getTotal()), the number of bytes already processed (getProcessed()), and the name of the file being processed (getFile()). Notice that, to improve performance, the number of bytes processed are not updated at every byte, but just at multiples of a predetermined value, given by the constant GRANULARITY.

For another thread to access these properties, it must have access to the MultipartRequest object that is processing the request. For this reason, each object has a unique ID returned by the method getID(). Given its ID, retrieving the MultipartRequest object through the method get(int) is possible. The ID is just an int value starting at 0 and incremented each time a new MultipartRequest object is created.

Besides parsing the request, the method setRequest(HttpServletRequest) also copies any and all parameters from the HttpServletRequest to the MultipartRequest object (see the method copyParameters(HttpServletRequest)). This takes advantage of any processing already done by the class HttpServlet so that you can also use the MultipartRequest API to access any fields present in the request's URL. As a side effect, this allows you to use the MultipartRequest class and common logic to process any kind of request.

The way files are saved is determined by the Multipart.File class. My implementation saves them as temporary files using the method java.io.File.createTempFile(String,String). For this reason, it is impor-

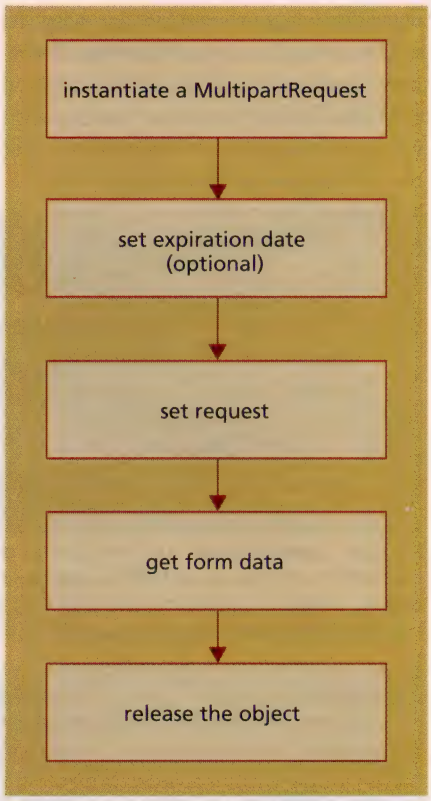


Figure 7 | Using the Multipart Class The simplest use of the MultipartRequest class (when you don't need to monitor the progress of an upload) is straightforward.

Method	Description
getInputStream()	Gets an input stream to read the contents of the file. The input stream returned by a previous call to getInputStream(), if any, is automatically closed.
getLength()	Returns the length of the file.
getName()	Gets the original file name, as sent with the request by the browser. Notice that the file name may contain path information as well, and depends on the client's platform.
getType()	Returns the MIME type of the file, as sent by the client, or an empty string if not known. This corresponds to the Content-Type header of the file part. Notice that, because MIME types are case insensitive, the type is always returned in lowercase.

Table 6 | MultipartRequest.File API The class MultipartRequest.File encapsulates any files uploaded by the request. Using appropriate methods, you can get the file contents and other properties.

tant to delete them when they are no longer needed to save space on the server. This is done by the method release(), which should be called on every MultipartRequest object after using it.

Each MultipartRequest object has an additional property, expiration date, which defines a limit date for the lifetime of that object. If the object is not explicitly released by calling its release() method, it is released automatically. This is done by the method releaseExpired(), which is called periodically by a background task, represented by the class MultipartRequest.ReleaseThread, created by the class MultipartRequest. The expiration date can be set and queried through the methods setExpiration() and getExpiration(), respectively. If not specified, the expiration date is set automatically depending on the object creation time and the value of the constant TIMEOUT.

This guarantees that all objects and temporary files are always released, even in the case of programming errors or conditions that prevent the release() method from being called. This happens particularly when creating MultipartRequest objects to be monitored.

Component Use

I designed the MultipartRequest class to be used both with servlets and with JSP in two cases. The simpler case is when you don't need to monitor the progress of the upload. This happens particularly when you are sure the files you will receive are small. In this case, all you need to do is to follow the steps outlined in Figure 7.

First, you instantiate a MultipartRequest object and set its expiration date if you feel the upload may take longer than the default. Then you set the request to the HttpServletRequest object your servlet or JSP page received as

Listing 2 Sample Servlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;



public class DisplayParameters
  extends HttpServlet
{
  public void doGet( HttpServletRequest req,
    HttpServletResponse resp )
    throws ServletException, IOException
    {
      doPost( req, resp );
    }

  public void doPost( HttpServletRequest req,
    HttpServletResponse resp )
    throws ServletException, IOException
  {
      MultipartRequest request =
        new MultipartRequest();
      request.setRequest( req );
      resp.setContentType( "text/html" );
      PrintWriter out = resp.getWriter();
      out.println( "<HTML>" );
      out.print( "<HEAD><TITLE>" );
      out.print( "MultipartRequest Parameters" );
      out.println("</TITLE></HEAD>" );
      out.println( "<BODY>" );
      out.println( "<TABLE BORDER=1>" );
      out.println( "<TR><TH>Parameter</TH>
        <TH>Values</TH></TR>" );
      Iterator iterator =
        request.getParameterNames();
      while ( iterator.hasNext() )
      {
        String name = (String)iterator.next();
        Object[] values =
          request.getParameterValues(name);
        out.println( "<TR>" );
        out.println( "<TD>" + name + "</TD>" );
        out.println( "<TD>" );
        if ( values.length > 1 )
          out.println ( "<TABLE BORDER=1
          WIDTH=100%>" );
        for ( int i=0; i<values.length; i++ )
        {
          Object value = values[i];
          if ( values.length > 1 )
            out.println( "<TR><TD>" );
          if ( value instanceof
            MultipartRequest.File )
          printFile
          ( out, (MultipartRequest.File)value );
          else
            out.print( "<PRE>" + value.toString() +
                "</PRE>" );
          if ( values.length > 1 )
            out.println( "</TD></TR>" );
        }
        if ( values.length > 1 )
          out.println( "</TABLE>" );
          out.println( "</TD>" );
          out.println( "</TR>" );
      }
      out.println( "</TABLE>" );
      out.println( "</BODY></HTML>" );
      request.release();
    }

  private void printFile( PrintWriter out,
    MultipartRequest.File file )
    throws IOException
  {
    out.println( "<TABLE>" );
    out.println( "<TR><TH>Name</TH><TD>" +
      file.getName() + "</TD></TR>" );
    out.println( "<TR><TH>Length</TH><TD>" +
      file.getLength() + "</TD></TR>" );
    out.println( "<TR><TH>Type</TH><TD>" +
      file.getType() + "</TD></TR>" );
    if ( file.getType().startsWith("text") )
    {
      out.println( "<TR><TH>Contents</TH><TD>" );
      out.print( "<PRE>" );
      InputStream in = file.getInputStream();
      int c;
      while ( (c=in.read()) != -1 )
        if ( (c == '<') ||
          (c == '>') ||
          (c == '"') ||
          (c == '\'') )
          out.print( "&#"+c+";" );
        else
          out.print( (char)c );
          out.print( "</PRE>" );
          out.println( "</TD></TR>" );
    }
    out.println( "</TABLE>" );
  }
}
```

This servlet uses the MultipartRequest class to display an HTML page with any parameters it receives. Notice how simple it is to set up a MultipartRequest object and access its properties.

Listing 3 Make a Multipart Request

```
<HTML>
<HEAD>
  <TITLE>Simple Example</TITLE>
</HEAD>


<BODY>

<%@ page language="java" import="MultipartRequest" %>
<jsp:useBean id="req" class="MultipartRequest">
  <jsp:setProperty name="req"
    property="request"
    value="<%= request %>" />
</jsp:useBean>

<%
  String type =
    req.getFileParameter("resume").getType();
  long length =
    req.getFileParameter("resume").getLength();
  if ( req.getStringParameter(
    "firstname").equals("") ||
```

Continued on page 47.

It is simple to set up a MultipartRequest object in a JSP page using the JSP's JavaBean support. The body of the useBean tag sets the request property of the MultipartRequest object just created to the implicit HttpServletRequest object (request) every JSP page receives.

Parameter	Values
submit-name	Larry
submit-file	**Name** C:\Netlog.txt **Length** 477 **Type** text/plain **Contents** NETDI: ClassInstall (0x6 on 0x1a16:0x58) on at NETDI: dif_FirstTimeSetup NETDI: SetDefaultTransport: Couldn't determine transport NETDI: ClassInstall(0x6) end NETDI: NETDI: lpRegLogConf = 0x0 NETDI: ClassInstall (0x9 on 0x1a16:0x58) on at NETDI: ClassInstall(0x9) end NETDI: ClassInstall (0xa on 0x1a16:0x58) on at NETDI: ChangeLine: No matches found. NETDI: ClassInstall(0xa) end NETDI: ClassInstall (0xc on 0x1a16:0x58) on at NETDI: ClassInstall(0xc) end

Figure 8 | Show What You've Got The DisplayParameters servlet returns a page conveniently formatted to display any parameters it received. Notice the properties of the file submitted as the submit-file parameter.

input, starting the upload. When the method setRequest(HttpServletRequest) returns, you can access the request's parameters, including files, performing any process you want. Finally, you call the release() method on the MultipartRequest object.

The sample servlet (DisplayParameters.java) uses the MultipartRequest class to display an HTML page with any parameters it receives (see Listing 2). If the parameter is a file, it displays its properties and contents if it has the "text" type. Notice the proper use of the MultipartRequest class and its API. Figure 8 shows the servlet output if you enter the data from Figure 1 in the page produced by Listing 1. (The mapping of URLs to servlets and JSP pages and the placement of files depend on the container, with which you must be familiar to configure it to work with the examples.)

Using the MultipartRequest class in a JSP page is as simple as in a servlet. From a user standpoint, the setup is even simpler, because MultipartRequest is modeled as a JavaBean, which is particularly appropriate for use with JSP pages. A simple JSP page (SimpleExample.jsp) is closer to a real application. It could be part of the job application section of a hypothetical company Web site (see Listing 3).

Listing 4 and Figure 9 show the input form (SimpleExample.html) used with this example. It accepts some basic information about the applicant and a resume, performs some validation, and outputs the data entered. Notice how the properties of the MultipartRequest.File object are used to validate the size and type of the resume. If the resume is not within the size and types expected, an error message is displayed.

The JSP page explicitly imports the MultipartRequest class using the import attribute of the page directive. This is important because the MultipartRequest class is not in any package—it is part of an unnamed package. This can cause a compilation error if the servlet generated from the JSP page uses a package, because the compiler may try to locate the class in the same package as the automatically generated servlet.

The MultipartRequest class is designed so that you can also display the progress of the upload. This is particularly useful when the files you expect to receive are large, taking several minutes to transfer, so that the users have visual feedback and are not wondering what is happening.

This case is more difficult than the first scenario presented. To do this, you must use a combination of server- and client-side code and several tricks that are much easier to understand by example. Start by taking a look at Listing 5 (UploadForm.jsp). It is a simple form with five file select controls (see Figure 10). At first glance, it looks like any form that uploads files, but there are some important differences.

First, UploadForm.jsp instantiates a MultipartRequest object. Then it encodes its ID as a parameter for the form's action attribute. Finally, it generates JavaScript code for the Upload button that opens a new window to display the status of the upload and submits the form. Unlike previous examples, the MultipartRequest object is instantiated when the form is displayed, rather than when it is submitted. This is necessary because its ID must be passed

> At first glance, UploadForm looks like any form that uploads files, but there are some important differences

Continued from page 46.

```
    req.getStringParameter("lastname").equals("") ||
    req.getStringParameter("email").equals("") ||
    (req.getStringParameter("sex") == null) ||
    (req.getStringParameter("position") == null) ||
    req.getFileParameter(
      "resume").getName().equals("") )
  {
    %>
    <B>
    There is a problem with your application!<P>
    You must enter your first and last names,
    email address, sex, position(s) you are
    interested and your resume.
    </B>
    <%
  }
  else if ( type.equalsIgnoreCase("text/plain") ||
    type.equalsIgnoreCase("text/richtext") ||
    type.equalsIgnoreCase("text/html") ||
    type.equalsIgnoreCase("application/msword") )

  {
    if ( length > 100*1024 )
    {
      %>
      <B>
      There is a problem with your resume!<P>
      We can't accept resumes larger than 100Kb.
      </B>
      <%
    }
    else
    {
      %>
      Your name:
        <%= req.getParameter("firstname") %>
        <%= req.getParameter("lastname") %><BR>
      Your email:
        <%= req.getParameter("email") %><BR>
      Your sex: <%= req.getParameter("sex") %><P>
      Position you are interested in:
        <UL>
      <%
        Object[] values =
          req.getParameterValues("position");
        for ( int i=0; i<values.length; i++ )
        {
          %>
          <LI><%= values[i] %></LI>
          <%
        }
      %>
      </UL>
      Additional information:<BR>
      <PRE><%= req.getParameter("notes") %></PRE><P>
      Your resume:
      <%= req.getFileParameter(
        "resume").getName() %>
      <%
    }
  }
  else
  {
    %>
    <B>
    There is a problem with your resume!<P>
    We can only accept unencoded/uncompressed
    Resumes as plain text, RTF, HTML or MS Word
    documents.
    </B>
    <%
  }
  req.release();
%>

</BODY>
</HTML>
```

Listing 4 Job Input

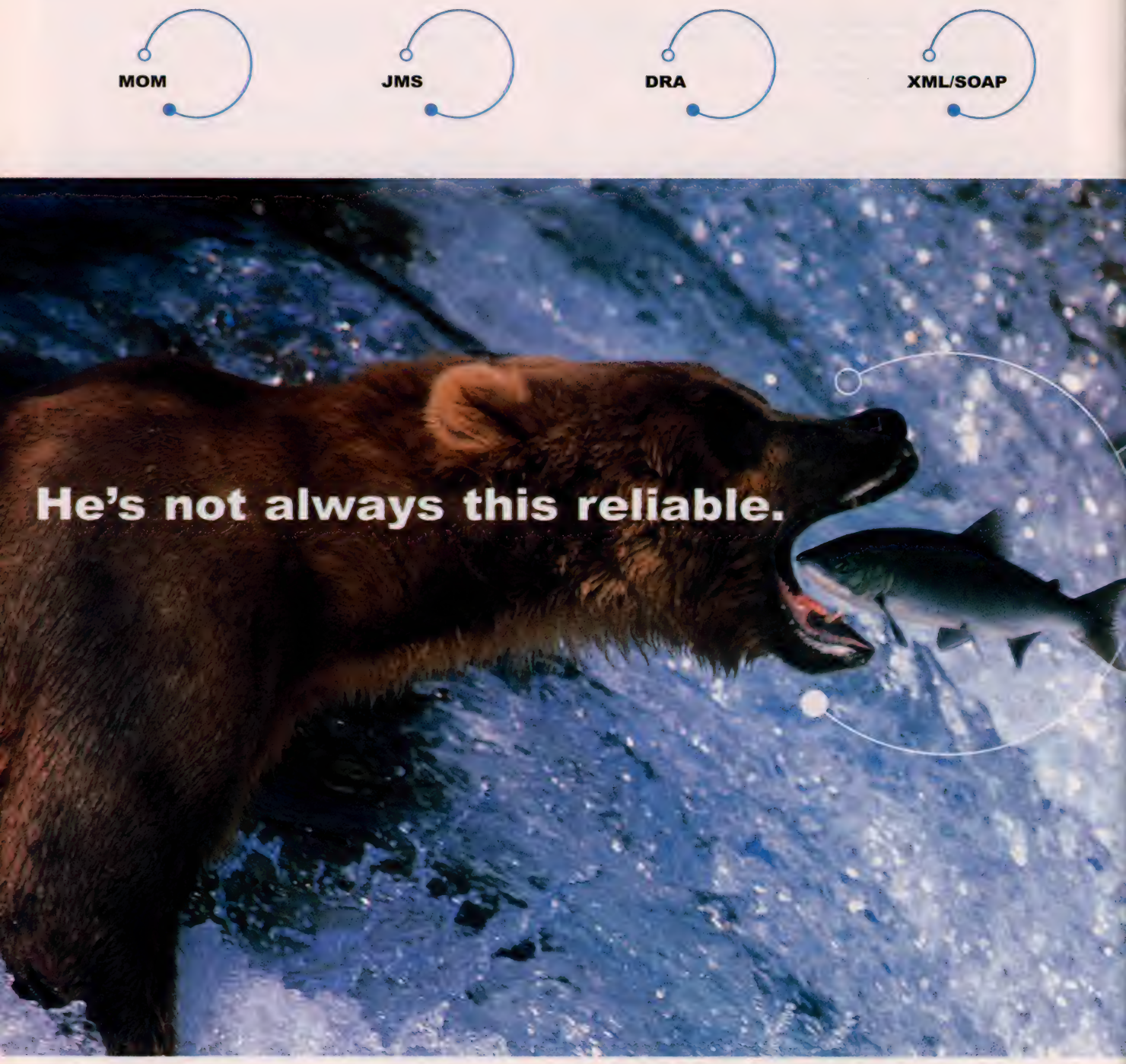
```
<HTML>
<HEAD>
  <TITLE>Simple Example</TITLE>
</HEAD>


<BODY>

<FORM
  action="SimpleExample.jsp"
  enctype="multipart/form-data"
  method=post>
  Name (First, Last):
  <INPUT type="text" name="firstname">
  <INPUT type="text" name="lastname"><BR>
  email: <INPUT type="text" name="email"><BR>
  Sex: <INPUT type="radio" name="sex"
    value="Male"> Male
  <INPUT type="radio" name="sex" value="Female">
    Female<BR>
  Position you are interest in
  (check all that apply):<BR>
  <SELECT multiple name="position">
  <OPTION value="Analyst">
    Systems Analyst</OPTION>
  <OPTION value="Engineer">
    Software Engineer</OPTION>
  <OPTION value="Architect">
    Systems Architect</OPTION>
  <OPTION value="Programmer">Programmer</OPTION>
    </SELECT><BR>
    Additional information (your availability,
      why you are interested in our company,
      etc):<BR>
    <TEXTAREA rows=5 cols=80 name="notes">
      </TEXTAREA><BR>
    Send your resume:
    <INPUT type="file" name="resume"><BR>
    <FONT SIZE=-1>
    <B>Important:</B>
    <UL>
    <LI>Plain text, RTF, HTML or
      MS Word documents only</LI>
    <LI>100Kb max</LI>
    <LI>Don't compress, archive or encode your
      resume</LI>
    </UL>
    </FONT>
    <INPUT type="submit" value="Send">
    <INPUT type="reset">
</FORM>

</BODY>
</HTML>
```

This is the source code for the job application JSP page example's form. It follows the guidelines for all forms that accept files; that is forms that use file select controls.



Java Compatible™
Enterprise Edition

© Copyright 2001 Sonic Software Corporation. All rights reserved. SonicMQ is a registered trademark and Dynamic Routing Architecture is a trademark of Sonic Software Corporation in the U.S. and other countries.

The big brown bear, *Ursus arctos*, of Kodiak Island, is an extraordinarily reliable hunter. Using its penetrating sense of smell and its lighting-quick reflexes, it detects and plucks spawning salmon from Alaska's wild waters with magical precision.

We are.

Worried about losing business-critical information you send over the Internet?

You should be— but not if you rely on SonicMQ®.

SonicMQ is the award-winning next-generation E-Business Messaging Server from Sonic Software. Unlike traditional message-oriented middleware products, **SonicMQ** is designed expressly for the Internet, and its patent-pending Dynamic Routing Architecture™ (DRA) incorporates durable subscriptions and persistent messaging to guarantee information delivery with absolutely no data loss. Plus, **SonicMQ** delivers the real-time performance and massive scalability required by e-business applications—while being exceptionally easy-to-use.

Find out more and learn why SonicMQ is being called the "de facto standard for JMS messaging." Visit www.sonicsoftware.com/jp.

sonic
SOFTWARE™
Backbone of the Extended Enterprise

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All other marks contained herein are the property of their respective owners.

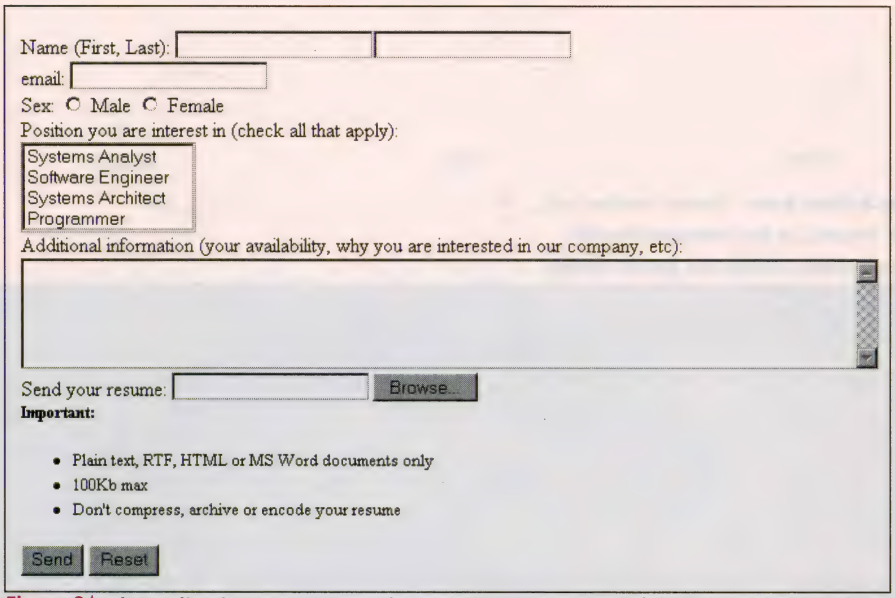


Figure 9 | Job Application Input Form The most important part of the job application form is the file select control that allows the applicant to submit his or her resume. Notice the requirements right below the control. They reflect the validation performed on the file.

both to the page that processes the form and to the page that monitors the transfer.

When the Upload button is clicked, its onclick event handler is executed, which submits the form to UploadResult.jsp (see Listing 6). This page processes the request, displaying the names of the files sent, and releases the MultipartRequest object, like in the previous examples. The difference is that the UploadResult.jsp page does not instantiate a MultipartRequest object to parse the request, because it was already instantiated by the UploadForm.jsp page when the form was displayed. All the UploadResult.jsp page needs to do is get a reference to the previously instantiated object. This is done using the MultipartRequest.get(int) method, which returns a reference to the object given its ID. That's why the ID of the MultipartRequest object was passed as part of the request's URL.

Listing 5 Monitor the Progress

```
<HTML>
<HEAD>
  <TITLE>Upload Status Example</TITLE>
</HEAD>


<BODY>

<%@ page language="java" import=
  "MultipartRequest" %>
<jsp:useBean id="req" class= "MultipartRequest" />

<FORM
  action="UploadResult.jsp?id=<%= req.getID() %>"
  enctype="multipart/form-data"
  method=post>
  Select the files you want to upload:<P>
  File 1: <INPUT type="file" name="file"><BR>
  File 2: <INPUT type="file" name="file"><BR>
  File 3: <INPUT type="file" name="file"><BR>
  File 4: <INPUT type="file" name="file"><BR>
  File 5: <INPUT type="file" name="file"><P>
  <INPUT type="button"
    value="Upload"
    onclick='window.open("UploadStatus.jsp?id=
    <%= req.getID() %>","status","height=100,
    width=400"); submit();'><P>
</FORM>

</BODY>
</HTML>
```

When you want to monitor the progress of file uploads, you must instantiate a MultipartRequest object before displaying the form, so that you can send its ID to both the result and status page/servlet.

Listing 6 Process the Form

```
<HTML>
<HEAD>
  <TITLE>Upload Status Example</TITLE>
</HEAD>


<BODY>

<%@ page language="java" import="MultipartRequest" %>
<%
  int id = Integer.parseInt(
    request.getParameter("id"));
  MultipartRequest req =
    MultipartRequest.get(id);
  req.setRequest( request );
  %>
  The following files were uploaded:
  <UL>
  <%
  Object[] values =
    req.getParameterValues("file");
  for ( int i=0; i<values.length; i++ )
  {
    MultipartRequest.File file =
      (MultipartRequest.File)values[i];
    if ( !file.getName().equals("") )
    {
      %>
      <LI><%= file.getName() %></LI>
      <%
    }
  }
  %>
  </UL>
  <%
  req.release();
%>

</BODY>
</HTML>
```

Notice how the reference to the previously created MultipartRequest object is obtained using the MultipartRequest.get(int) method's ID, extracted from the request's URL through the ServletRequest.getParameter(String) method.

This way it is automatically parsed and made available through the HttpServletRequest object (in the case of a JSP page, that's the implicit "request" object) as a parameter, independent of anything else.

Besides submitting the form, the Upload button also opens a new window to display the page UploadStatus.jsp (available online). This is the page that displays the status of the upload so that the user can see its progress. First it gets a reference to the same object instantiated by the page UploadForm.jsp. This is the object that the page UploadResult.jsp will also use to parse the request. Once it gets its reference, UploadStatus.jsp can use its API (getTotal(), getProcessed(), and getFile()) to get the status of the upload.

In this instance, the code has to handle a couple of cases. If the method get(int) returns null, that means the request was already processed and the MultipartRequest object was released, so UploadStatus.jsp closes the window. Otherwise, if the method getTotal() returns 0, that means the request hadn't started being parsed yet. In this case, the other methods didn't return meaningful results. If getTotal() is not 0, than the other methods are called to get the status and display it as raw data (see Figure 11). UploadStatus.jsp uses JavaScript code to refresh the window every five seconds, so the page is redisplayed updating the status information.

Although the example uses JSP pages and windows and displays the status as raw data, you can use servlets and frames and display the data as a progress bar, or something else, or calculate additional information.

This example is not bulletproof. For instance, if the user cancels the upload using the browser's Stop button and tries to submit it again, it won't work. The code does not allow the MultipartRequest object to be reused. The same way, if the user uses the browser's Back button to go back to the form after uploading a file, the browser doesn't redisplay the form, which would instantiate a new object, but uses the same one instead, leading to the same problem.

Additionally, if the user displays the form and doesn't submit it, a MultipartRequest object will be instantiated but not used. In this case, the UploadResult.jsp page will not be called to release the object, so it will be automatically released by the ReleaseThread thread after its expiration date. This isn't a problem, but if the user takes too long to submit the form and submits it after the object's expiration date, the UploadResult.jsp and UploadStatus.jsp pages will get a null reference when trying to retrieve the MultipartRequest object because it was already automatically released.

All of these issues can be prevented using client-side scripting code or testing for these conditions in the code. I chose not to implement this logic to keep the example simple, but a real-world application should include code for these situations.

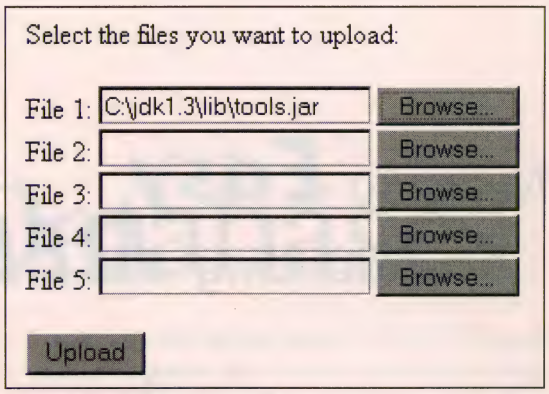


Figure 10 | Make Your Selection UploadForm.jsp contains just five select controls, but more exist behind the scenes.

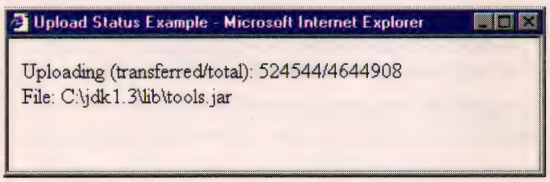

Upload Status Example - Microsoft Internet Explorer
Uploading (transferred/total): 524544/4644908
File: C:\jdk1.3\lib\tools.jar

Figure 11 | Upload Status UploadStatus.jsp refreshes the status window every five seconds.

Restrictions and Limitations

My implementation of multipart/form-data doesn't support the full standard—just a subset of it. Regarding MIME entity headers, this example doesn't support character escaping, header wrapping, comments, or encoded non-ascii characters. It assumes that parameters are sent using the western (iso-8859-1) character set. It doesn't support the charset attribute of the Content-Type header. In addition, it doesn't support encoded parts (Content-Transfer-Encoding header) or multipart/mixed parts (used to transfer multiple files as a single parameter).

Although these features are part of the standard, neither Netscape nor Internet Explorer uses them. So I decided to leave them out to simplify the code.

Finally, I did no error checking regarding the syntax of the request. Because it is machine generated and the HTTP/TCP protocol guarantees that the data is not corrupted, it is safe to assume that the request is valid and well formed. The only error that can occur is if the connection is broken, but this will cause an IOException. Removing these tests from the code makes it much smaller and easier to understand.

Although Java provides decent support and an array of solutions for server-side programming, it still lacks some features. However, with a bit of effort, you can take advantage of this support and build new features on top of its existing solutions, without reinventing the wheel.

Now you know how to add support for a new data format, multipart/form-data, using existing logic from both HttpServlet and HttpServletRequest. The result is a new class that fits well in the servlet model, extending its capabilities. JP

Although the example uses JSP pages and windows, and displays the status as raw data, you can use servlets and frames and display the data as a progress bar

GO ONLINE
www.javapro.com

Use these Locator+ codes at www.javapro.com to go directly to these related resources.

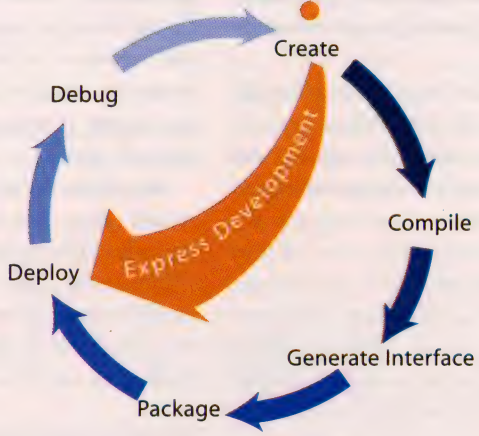
JP0201 Download all of the code for this issue of *Java Pro*.

JP0201AC Download the code for this article separately.

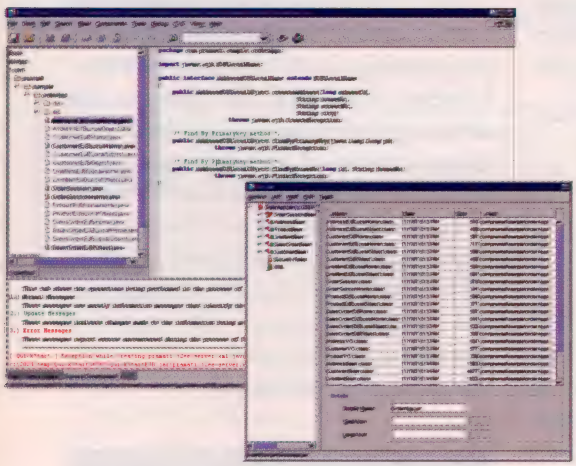
JP0201AC_T Read this article online.

PRAMATI

Never So **Easy** Building **J2EE™ 1.3** Systems



Pramati Studio 3.0



- Create-Deploy Full J2EE 1.3 Apps via Express Development
- Generate EJB 2.0 Code, Including CMRs and MDBs
- Full Compliance via High Fidelity J2EE Methodology Mapping
- Migrate Sources From Any Platform or Export as Standard Archives
- In-built Pramati Server for Desktop Deployment, Testing and Debugging
- Plug in Any Application Server as Development Platform
- Debug JSP Scripts, Step-in to EJB Methods, Remote Debugging
- Use ANT™ Scripts for Building Complete Projects
- Integrated CVS, Framework for Plugging in Other Source Control Systems
- Pure Java Tools Harness to Carry Highly Specialized Toolset

Download

60 Day Free Evaluation

Pramati Studio 3.0

Pramati Server 3.0

Pramati Message Server 3.0

For more information contact *info@pramati.com* or visit *www.pramati.com*

San Jose New York Hyderabad Bangalore

Java, Enterprise JavaBeans and JavaServer Pages are the trademarks of Sun Microsystems in the USA and other countries. ANT is a registered Trademark of Apache Software Foundation

Sophisticated Databases

Getting the best performance from your database calls requires that you plan ahead

Databases are the bread and butter of virtually any Web application that serves customers, and they are the basis of any material contribution to a business entity. Just as developers have used tools such as Visual Basic to build front ends to databases for traditional client/server applications, Java programmers today use JavaServer Pages (JSP) to create Web-based user interfaces for manipulating data in databases.

Thanks to the versatility of JSP technology, it's not difficult to do so. In "Calling All Databases" (Weblication column, December 2001), we made a simple database call from a JSP page (SELECT * FROM Products) using the SQL language, and brought the data back into a resultset in the JSP page. Then we copied the data from the resultset into variables in preparation for display on the page. In the next step, we looped through the resultset to pull the data out of that structure. The last step in the process is to ready it for display.

Once the data is extracted from the resultset, displaying retrieved data on the page is a simple process. One way of doing it is to use the print method of the out object, which writes the value out to the page. Assigning data to a string variable affords you some flexibility in formatting, but it is not required. You'll normally use variable types for output that correspond to the data you query:

```
while putData.next( )
{
  String ProductName =
    putData.getString("prod_name");
  out.print(ProductName);
}
```

Before you write code to get data from the database, it's important to design your output display. When designing the data output, you should have a picture in your mind, or sketched on a piece of paper, of the page after it has been filled in with data, and implement that picture using HTML. If "human factors" expertise isn't in your repertoire of skills, you should consult a design professional or at least try different page mockups on several target users.

Thoughtful page design not only improves the visual appearance of the data, but also makes for a better user experience by enabling the application to seem faster than it really is. Database calls can be slow to execute and return data, and by designing your page and your code to minimize the user impact of database calls, you'll improve the overall usability of the application.

Often, it makes sense to take a performance hit once, rather than several times. For example, you may not need to output all of the data from the resultset, nor output it in the same order as it came from the database. Although this might seem odd, a moment's reflection can provide several reasons for limiting or rearranging the output. One reason is that you may want to filter or process data further in a way that might be computationally expensive or inconvenient in the SQL calls. For example, the data may need to be sorted, from lowest to highest price, or you may want to pull out only the 10 most expensive items for display. Doing this on the page can make sense, especially if you anticipate that the user will request further manipulations on the same data. One advantage of doing so is that once you've made the call to the database, you've taken the performance hit associated with running the query.

You can also make these adjustments with script on the page itself. If you use the flush method associated with the out object, you can display any static text and graphics on the page before the database query returns. This prevents the user from having to look at a blank page for a long time—a situation that makes a database call seem painfully slow in practice.

Add and Update Records

You can add a record or update a record in a database from within a JSP page, either through the ResultSet object or by using SQL. To work with the resultset you've already defined, it's a matter of adding a new row. The updated resultset will be reflected back automatically into the database

by Peter VARHOL

Peter Varhol is a technical evangelist for Compuware Corporation. You can reach him at **Peter.Varhol@compuware.com**.

Thoughtful page design not only improves the visual appearance of the data, but also makes for a better user experience by enabling the application to seem faster than it really is

through the use of the autocommit feature of the Connection object.

To do this, the resultset you've retrieved has to include all of the data fields that you want to write back into the database. In other words, if you've retrieved only the product name, you can't write back a new product name, product ID, and quantity, simply because the resultset doesn't have a place to put the additional fields. Once you're sure that the resultset is appropriate, you move the file pointer, or cursor, down to the row into which you want to make your insertion. This is done through the moveToInsertRow() method of the ResultSet object:

```
ResultSet putData =
  getDava.executeQuery(
  "SELECT * FROM Products");
PutData.moveToInsertRow( );
```

Once the cursor is set on the update row, you can use several possible update methods to write data into the resultset, depending on the type of data being written. Two of the most common are UpdateInt(), for writing integer values, and UpdateString(), for writing string values. Whichever update method you use requires that you pass the column into which you want to insert the data, and the data value to be inserted.

One other necessary modification that you need to make to change the resultset is to set the resultset type and concurrency type. The resultset can be one of three types: TYPE_FORWARD_ONLY, TYPE_SCROLL_INSENSITIVE, or TYPE_SCROLL-SENSITIVE. Either of the last two lets you change the resultset and have it update the database; the only difference is that TYPE_SCROLL-SENSITIVE updates the database immediately, but TYPE_SCROLL_INSENSITIVE waits until you close the connection. The concurrency type can have the values CONCUR_READ_ONLY or CONCUR-UPDATABLE, of which the latter is necessary to change the resultset:

```
Statement getData =
  myConnection.createStatement(
  ResultSet. TYPE_SCROLL-
  SENSITIVE, ResultSet. CONCUR-
  UPDATABLE);
PutData.updateString(
  1, "Watchband");
```

The first parameter in the updateString() method refers to the first field in the resultset data. You can also refer to it by field name, which in this case might be product_name.

Last, you perform the insertion with the insertRow() method of the ResultSet object:

```
PutData.insertRow( );
```

Alternatively, you can execute a SQL statement directly from the script by using the ExecuteUpdate() method of the Statement object. It could also help if you assign your SQL statement to a variable, so that the statement itself can be abstracted from the update instruction:

```
String SQLcall =
  "INSERT into Products (
  1, "Watchband");
GetData.executeUpdate(SQLcall);
```

However, getting data into the database is only half the problem. Typically, a user will enter data, such as contact information to join a mailing list, from a Web form. To access data entered into a form, you can use the getParameter() method of the Request object. For example:

```
Request.getParameter(
  "product_name");
```

Updating an existing record is a similar process. As when adding a record, you set the cursor to the appropriate line in the resultset, and use one of the update methods of the ResultSet object. Then, instead of the insertRow() method, you make use of the updateRow() method of ResultSet:

```
PutData.updateRow( );
```

You can also make SQL calls, in this case using the UPDATE instruction, directly from the script to perform this action:

```
GetData.executeUpdate(
  "INSERT into Products (
  1, "Watchband"));
```

In many cases, you will want to hold off on committing changes to the underlying database until you're able to write several changes at once. That's because calls to a database can be computationally expensive, so it makes sense to group them when possible. To accomplish this, first turn off the database connection's autocommit feature using the setAutoCommit() method:

```
myConnection.setAutoCommit(false);
```

To execute multiple SQL statements, you add the statements by using the addBatch() method of the Statement object:

```
GetData.addBatch(
  "INSERT into Products (
  1, "Watchband"));
GetData.addBatch(
  "INSERT into Products (
  1, "Bracelet"));
GetData.addBatch(
  "INSERT into Products (
  1, "Ring"));
```

Once all of the addBatch() methods have been included, you have to execute the batch with the executeBatch() method and commit the changes to the database. Last, if that represents the conclusion of your batch update needs for the time being, you'll want to set the autocommit feature back to true:

```
GetData.executeBatch( );
myConnection.commit( );
myConnection.setAutoCommit(true);
```

If you have a small database and few users, optimizing your calls, querying the

right amount of data, and displaying your page before the query is complete probably aren't necessary, and actions such as these may seem like a lot of trouble for little gain. But if you want your JSP to scale to larger databases and more users, these actions become an absolute necessity. Seeking out ways to improve the performance of database queries will ultimately improve the user experience and make the application more reliable.

Prepared statements also speed up database access. These are SQL calls that are precompiled to save the compilation process during use. There's no need to do this if the statement will be executed only a single time, but if the statement will be run multiple times, using prepared statements makes sense.

Prepared statements are defined through the PreparedStatement object, which sends a SQL statement to the database manager. The PreparedStatement object is created through the Connection object using its prepareStatement() method and takes as its parameter the statement you want to precompile. You send the PreparedStatement object to the database by using its execute(), executeQuery(), or executeUpdate() method, depending on the function of the prepared statement:

```
PreparedStatement ps =
  myConnection.prepareStatement(
  "INSERT into Products (
  1, "Watchband"));
```

If the prepared statement you devise selects data from the database, you'll also have to write the resultset for the data returned from the query. That's simply a matter of creating a resultset similar to the one you're already working with:

```
While (PutData.next( ))
{
  String ProductName =
    putData.getString(
    "prod_name");
  out.print(ProductName);
}
```

These operations are enough to get started querying, updating, and adding to your database. As your database gets more sophisticated, you'll discover more methods to accomplish data queries and manipulations and tricks that enable you to get the best performance out of your database calls while hiding any performance issues from the user. The best method of learning both good user interface design and good coding practices is to experiment extensively and determine what works best for both your application and your data. JP

GO ONLINE
www.javapro.com

Use this Locator+ code at www.javapro.com to go directly to this related resource.

JP0201WC_T Read this article online.

Choosing a J2EE Application Server?

Independent J2EE Technology Evaluation

CSIRO Australia is working with product vendors to produce detailed, independent evaluations of the major J2EE technologies including:

★ BEA WebLogic ★ Barland Application Server ★
★ Silverstream Application Server ★ JBoss ★
★ Fujitsu INTERSTAGE Application Server ★
★ Websphere Application Server ★

The report is required reading for software architects, project managers and developers who need to thoroughly understand different application server technologies.

The report contains unique insights into the architectures, features, performance and scalability of these technologies. Each product has been used to develop a typical 3-tier business application, as the basis for comparing key features such as:

★ Performance ★ System management ★
★ Security ★ Database Integration ★ Scalability ★
★ Directory Services ★ Ease of development ★
★ Transactions ★ System management ★
★ Architecture and standards issues ★

Order "Evaluating J2EE Application Servers" online at:
www.publish.csiro.au (Choose computer science)
or
www.cutter.com/itgroup/reports/8reports.html

For more information visit:
www.cmis.csiro.au/sact

CSIRO

Take the shortcut.

Don't Get Stuck in the GUI Thread

A tutorial for avoiding some common perils and pitfalls in the Java GUI

by Charles W. **KANN**

Dr. Kann is an associate professor of computer science at University of Maryland, University College. He has worked in the computer industry for over 20 years, and has a PhD in computer science from George Washington University. His interests include how to effectively teach object-oriented, concurrent, and distributed programming, for which the Java language and architecture is a particular effective platform. You can e-mail him directly at **ckann@home.com**.

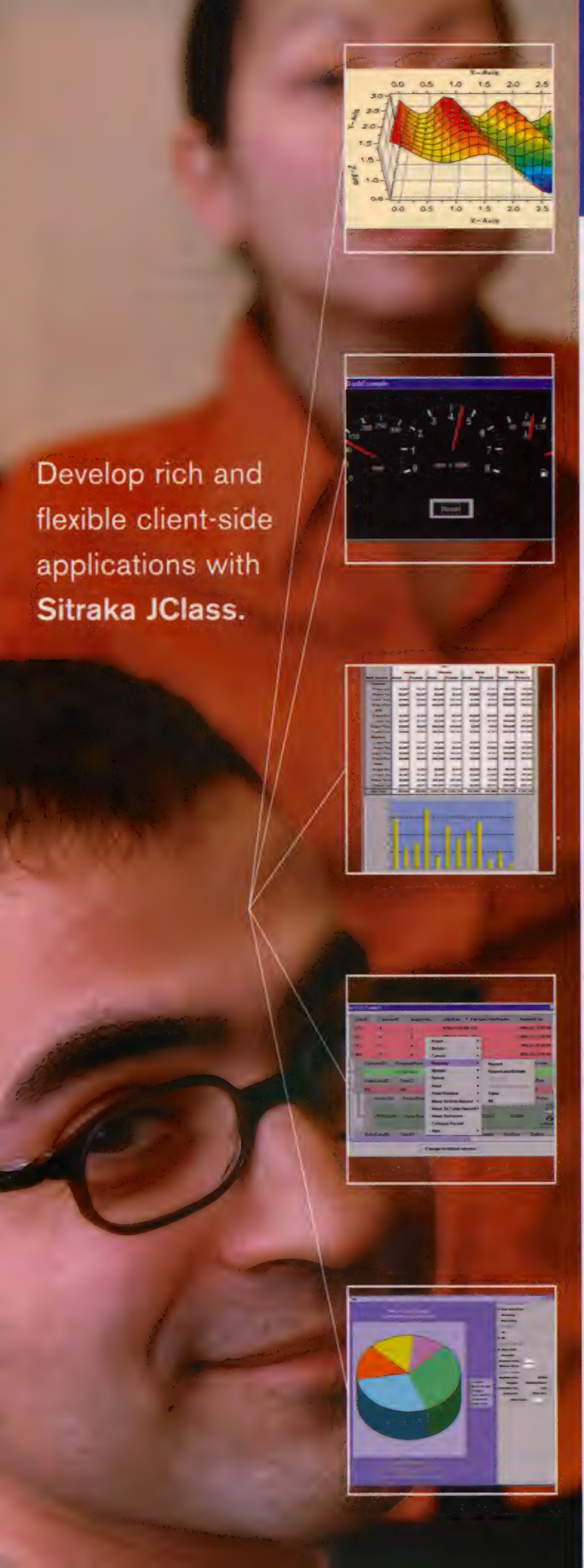
Most of my students have never done concurrent programming in Java or used Java threads, so I start out by putting the program shown in Listing 1 on an overhead. I use this program to teach concurrent programming for two reasons. The first reason is that many students do not believe threads are useful. This program shows that threads are not only useful, but are pervasive, if hidden, in modern programming environments. The second reason is to make them aware that a GUI thread is present when they use the Java AWT. Because this thread is present, it is important that programs using the AWT account for synchronization and deadlock, as in any concurrent program. But being aware of the presence of the GUI thread is even more important because if it is used incorrectly, a program can easily become stuck in the GUI thread, which can shut down the interface.

> The laws of magic might be an expedient explanation for students learning a new language, but of course it is always best to explain why a program works as it does

The program shown in Listing 1 has a loop that calculates a Fibbinocci series. The calculation loop can exit only when the value of the stopProgram variable is set to true. This variable is initialized to false when the program begins, and it is not changed anywhere in the body of the main function or any function called from main. Because the value does not change, students readily agree that the loop calculating the Fibbinocci series is an infinite loop and cannot be stopped from the program.

We then run the program, which creates a graphical user interface (GUI) for displaying the results of the Fibbinocci series and then starts the calculation of the Fibbinocci series. The GUI displays a button—when the button is pushed, the program stops. The students are always quick to point out that the listener for the button sets the value of stopProgram to true. But this violates the infinite loop in the program. Some students admit to being troubled by this behavior. Others have simply come to believe that computers obey the laws of magic, and some things, such as button listeners being called, can just happen, regardless of whether or not they can be explained.

The laws of magic might be an expedient explanation for students learning a new language, but of course it is always best to be able to explain why a program works as it does. For example, the mechanism that allows the button to work involves threads. When a GUI is created in a Java application, at least two threads are present. One thread runs through the code in the main() method and is called the main thread. The other thread controls the GUI and is called the GUI thread. When more than one thread is present in a program, they run *asynchronously*, that is, they both run with a separate thread



sitraka

JClass™

The world's leading Java components

JClass Enterprise Suite

Proven in enterprise environments around the globe, JClass Enterprise Suite includes all the functionality you need to build powerful, professional interfaces. Add to that unrivaled JDK, platform and IDE support, and you're looking at one exciting solution.

Includes all nine award-winning JClass components:

- **JClass** Chart (also available separately)
- **JClass** PageLayout
- **JClass** Chart 3D
- **JClass** LiveTable
- **JClass** Elements
- **JClass** JarMaster
- **JClass** Field
- **JClass** HiGrid
- **JClass** DataSource

JClass Chart

Embed the power of professional charts in your GUIs.

JClass Chart gives you the power to create sophisticated, interactive charts quickly and easily. This data-aware Java component is loaded with popular business and scientific chart types, which you can populate with data from a variety of sources including XML.

sitraka

the Java™ advantage

Download a free eval now

www.sitraka.com/jclass/jp

1 888-361-3264 | Europe +31(0)20 510 67 00

of control, as if they were separate programs. Thus the GUI thread appears to be running the button listener at the same time the main thread runs the calculation loop, and the GUI thread can change the value of the stopProgram variable without specifically being called in the main thread.

The paint() Method

You should remember two important things about the GUI thread. The first is that it is independent of the main thread, as was shown in the example in Listing 1. The second is that it will call only a limited number of methods. Some of these are standard methods, such as update() and paint(), and others can be specified (for example, listeners on graphic components such as buttons).

Listing 2 shows how the GUI thread works with the paint() and repaint() method. This simple animation program

Listing 1 Unexpected Results

```
import java.lang.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;

public class fib
{
  private static boolean stopProgram = false;

  public static void main(String argv[]) {
    Frame myFrame = new Frame(
      "Calculate Fibinocci Numbers");
    List myList = new List(4);
    myFrame.add(myList, BorderLayout.CENTER);
    Button b1 = new Button("Stop Calculation");
    b1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        stopProgram = true;
      }
    });
    myFrame.add(b1, BorderLayout.SOUTH);
    myFrame.setSize(200, 300);
    myFrame.show();

    int counter = 2;
    while(true) {
      if (stopProgram)
        break;
      counter +=1;
      myList.add("Num = " + counter + "  Fib = " +
        fibonacci(counter));
      myFrame.show();
    }
    // Note stopProgram cannot change value to
    // true in the above loop.  How does the
    // program get to this point?
    myList.add("Program Done");
  }

  public static int fibonacci(int NI) {
    if (NI <= 1) return 1;
    return fibonacci(NI - 1) + fibonacci(NI - 2);
  }
}
```

This program illustrates the benefits of concurrent programming. Although the code makes it look like the loop calculating the Fibbinocci series never exits, the result turns out differently than expected.

Listing 2 Too Much Sleep

```
import java.awt.*;
import java.awt.event.*;

public class SimpleAnimator extends Canvas {

  private int currentX = 0, currentY = 0,
    xLimit= 500, yLimit = 500;
  private static int sleepTime = 100;

  public static void main(String args[]) {
    Frame myFrame = new Frame();
    Panel controlPanel = new Panel();

    Button exitButton = new Button("Exit");
    exitButton.addActionListener(
      new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        System.exit(0);
      }
    });
    controlPanel.add(exitButton);

    controlPanel.add(new Label(
      "Animator Speed"));
    final Scrollbar speedControl =
      new Scrollbar(Scrollbar.HORIZONTAL,
      9900, 25, 10, 10000);
    speedControl.addAdjustmentListener(
      new AdjustmentListener() {
      public void adjustmentValueChanged(
        AdjustmentEvent e) {
        sleepTime = 10000 - e.getValue();
      }
    });
    controlPanel.add(speedControl);
    myFrame.add(controlPanel,
      BorderLayout.SOUTH);

    SimpleAnimator sa = new SimpleAnimator();
    myFrame.add(sa, BorderLayout.CENTER);

    myFrame.setSize(500,500);
    myFrame.show();
  }

  public void paint(Graphics g) {
    g.setColor(Color.red);
    g.fillOval(currentX, currentY, 15, 15);

    currentX = (currentX + 5);
    currentY = (currentY + 5);
    if (currentX > xLimit)
      currentX = 0;
    if (currentY  > yLimit)
      currentY = 0;

    try {
      Thread.sleep(sleepTime);
      repaint();
    }catch (InterruptedException e) {
    }
  }
}
```

This animation program paints a ball as it falls from the top left to the bottom right corner of the screen. The ball is always repainted slightly offset from the last position. The problem with this program is that the GUI gets stuck sleeping in the paint() method.

paints a ball as it falls from the top left to the bottom right corner of the screen. The animation effect is produced by calling the paint() method many times, always repainting the ball offset slightly from the last position. The program also has a scroll bar to control the speed of the ball and an Exit button to end the program.

The first concept to consider in this program is how best to achieve multiple calls to the paint() method. The mechanism for creating multiple calls is shown here:

```
public void paint(Graphics g) {
  //redraw something
  try {
    Thread.sleep(sleepTime);
    repaint();
  }catch (
    InterruptedException e) {
  }
}
```

Because most students cannot yet handle the concept of a multithreaded program, they believe that this code works by the repaint() method calling the paint() method through some unexplained (magic) mechanism, thus allowing paint() to be called multiple times. This explanation is insufficient for three reasons. First, how does the initial call to the paint() method occur? Second, if the repaint() method calls the paint() method, the repaint() method should pass the Graphics object g so that it can be used on subsequent calls to paint(). Third, if repaint() calls paint() then infinite recursion is happening, which should eventually cause a stack overflow.

Because recursion fails to explain the behavior of this program, some other mechanism must be at work. That mechanism involves the GUI thread. The GUI thread can be thought of as a big loop that always

Listing 3 Not a Good Listener

```
import java.awt.*;
import java.awt.event.*;

public class StickyButton {
  static boolean stopProgram = false;

  public static int fibonacci(int NI) {
    if (NI <= 1) return 1;
    return fibonacci(NI - 1) + fibonacci(NI - 2);
  }

  public static void main(String args[]) {
    Frame myFrame = new Frame();
    myFrame.setLayout(new FlowLayout());

    Button startButton = new Button("Start");
    startButton.addActionListener(
      new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        int count = 2;
        while (true) {
          if (stopProgram)
            break;
          System.out.println(fibonacci(count));
          count++;
        }
        System.out.println("Button completed");
      }
    });
    myFrame.add(startButton);

    Button stopButton = new Button("Stop");
    stopButton.addActionListener(
      new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        System.out.println("In Stop");
        stopProgram = true;
      }
    });
    myFrame.add(stopButton);

    myFrame.setSize(100,100);
    myFrame.show();
  }
}
```

In this Fibbinocci program, we have added a Start button that allows the user to control when the application starts. We have also moved the calculation loop into the GUI thread by putting it in the listener. The problem with this program is that the GUI thread gets suspended inside of the button listener, effectively disabling the Stop button.

Listing 4 Comparing Applets to Oranges

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class StickyApplet extends Applet {
  private boolean stopProgram = false;

    public int fibonacci(int NI) {
      if (NI <= 1) return 1;
      return fibonacci(NI - 1) + fibonacci(
        NI - 2);
    }

  public void init() {
    Button b1 = new Button("Stop Calculation");
    b1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        stopProgram = true;
      }
    });
    add(b1);

    int counter = 2;
    while(true) {
      if (stopProgram)
        break;
      System.out.println("Num = " + counter +
        "  Fib = " +
        fibonacci(counter));
      counter++;
    }
    System.out.println("Program Done");
  }
}
```

The GUI thread can get suspended in an applet also, particularly in the init() method. This code calculates the Fibbinocci series, but does so using the init() method. However, the Fibbinocci calculation is now being done in the GUI thread, not a separate main thread. The init() method never returns, so the interface never can show any of the graphic components.

runs, and calls methods based on the value of events that have been set since the last time the loop was run. When a call is made to the show() method for the frame, a "paint" event for the frame is scheduled. The GUI thread now starts running, finds a "paint" event, and calls the paint() method. When the repaint() method is called in the paint() method, it sets an event for the GUI thread to call the paint() method the next time through the loop. When the GUI thread returns from the paint() method, it runs the loop again, sees that a "paint" event has been set, and calls paint() again.

You could argue that although the explanation using recursion is technically incorrect, it explains the behavior of the repaint() method, and so is sufficient for understanding how repaint() works. The problem is that the explanation using recursion is not simply incorrect, but is insufficient to explain all the possible behavior when using the paint() method. Remember that the paint() method is called from the GUI thread. This is important because *all* graphic components are called from the GUI thread, including the button and scroll bar listeners. Note that inside of the paint() method, a call to Thread.sleep is used to control the speed of the animation. Because this call to Thread.sleep occurs in paint(), the GUI thread does not return from the paint() method until the call to sleep() is finished, thus suspending the GUI thread while sleep() executes. Because all the other graphic components are also handled by the GUI thread, they are also suspended, and events to the button and scroll bar cannot be processed until the call to sleep() completes. To see this effect, run the code shown in Listing 2, pull the scrollbar all the way to the left, and then try to readjust the scroll bar or press the button. You will see a lag in accessing these components of up to 10 seconds—the 10 seconds the GUI is sleeping in the paint() method. The interface has become stuck in the GUI thread.

Sticky Buttons

To illustrate this stickiness in the GUI further, consider the program in Listing 3. It is the same Fibbinocci program from Listing 1, but a Start button has been added that allows the user to control when the application starts. Now the calculations start when the button is pressed. However, this program is different from the one in Listing 1 in that the calculation loop is no longer in the main thread but has been moved into the GUI thread by putting the calculation loop in the listener. This has the unintended consequence of suspending the GUI thread inside of the button listener so that it can no longer look for events from other buttons. The Stop button has effectively been disabled.

This type of situation can arise easily in the design of a user interface. Consider this scenario: You write an application that gathers data and submits it to a remote server. The application has two buttons—Submit, which sends the information to the server, and Cancel, which interrupts the submission of the data. If the call to the remote server is done synchronously in the listener for the Submit button, the GUI thread will not enter the listener for the Cancel button until the submit is finished, making the Cancel button effectively useless.

This highlights a rule that you should follow when working with methods such as paint(), or component listeners that are called in the GUI thread. Only attempt short activities in the GUI thread. Anything that will take more than a short period of time should be done in another thread.

> The presence of these threads cannot always safely be ignored, and so it is incumbent on the programmer at least to understand that these threads are present, and to make allowances for unsafe interactions that occur when using them

Sticky Applet

The problem with becoming stuck in the GUI thread also exists in applets. In an applet, the init() method runs in the GUI thread. Because many applets start in the init() method, it is easy to become stuck. For example, consider the program in Listing 4. Here the init() thread is calculating the Fibbinocci series, just as was done in the main thread in Listing 1. Here, however, the Fibbinocci calculation is being done in the GUI thread, not a separate main thread. Because the init() method never returns, this program represents a true infinite loop, because the GUI thread can never process other events. This means that the interface can never show any of the graphics components.

This problem can be fixed easily by putting the Fibbinocci calculation in a separate thread (see Listing 5 online at www.javapro.com). This program now has two threads: one for calculations, and one for the GUI (as in the program in Listing 1). In this example, the explicitly created thread is equivalent to the main thread, because it is running separately from the GUI thread.

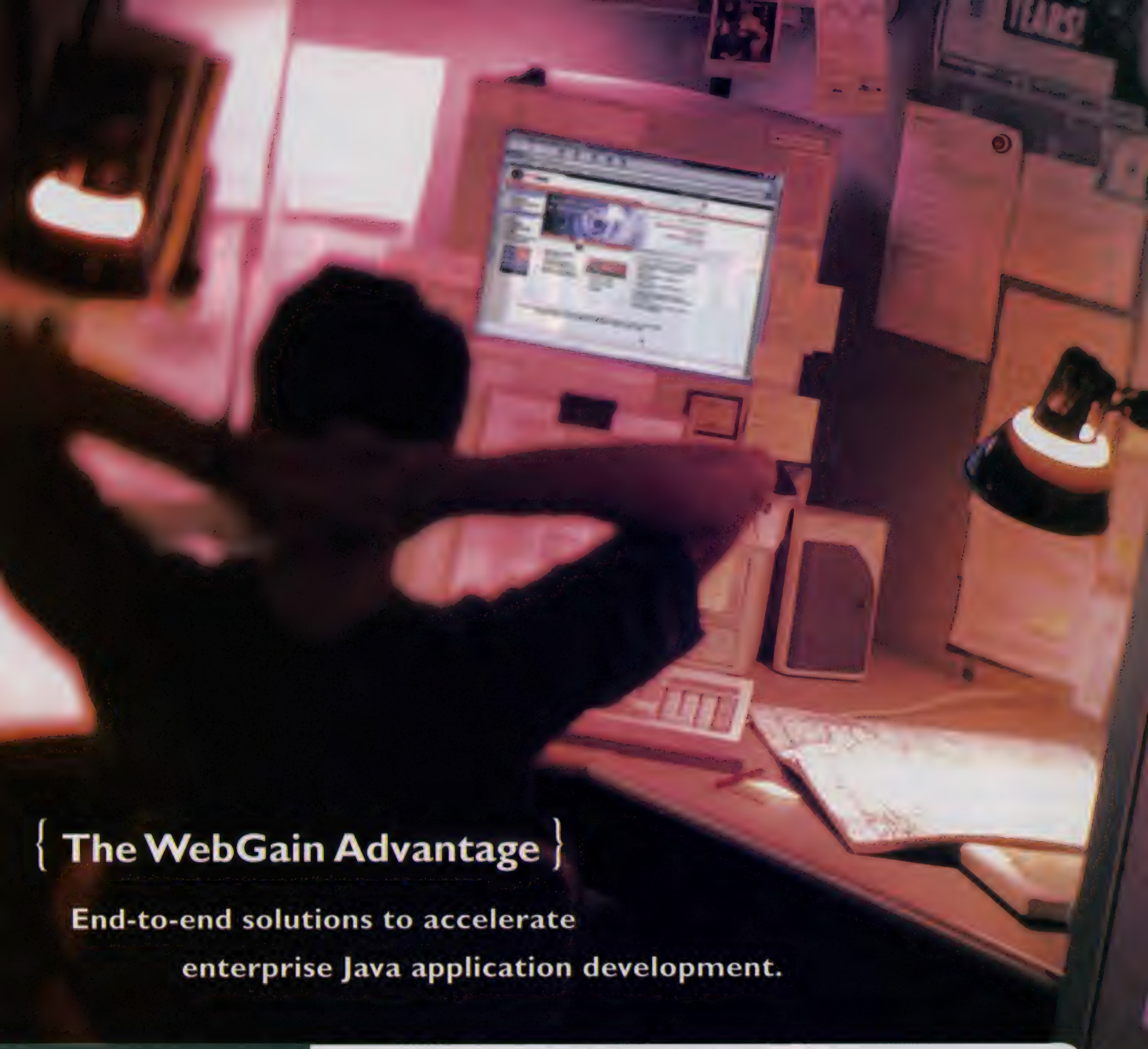
Threads are common in Java, even if they are hidden. Every time programmers use the Java AWT, they create threads. The presence of these threads cannot always safely be ignored, and so it is incumbent on the programmer at least to understand that these threads are present and to make allowances for unsafe interactions that occur when using them. Not understanding these threads could lead a programmer to implement a program that seems perfectly rational, but has unexpected, and unwelcome, behavior. JP

Use these Locator+ codes at www.javapro.com to go directly to these related resources.

JP0201 Download all of the code for this issue of *Java Pro*.

JP0201CK Download the code for this article separately.

JP0201CK_T Read this article online.



{ The WebGain Advantage }

End-to-end solutions to accelerate enterprise Java application development.

WebGain Studio™	Integrated Development Suite
VisualCafé™	Accelerate Java Development
TopLink™	Object to Relational Data Integration
StructureBuilder™	Synchronize UML and Java
Quality Analyzer™	Build Quality Applications
Business Designer™	Manage Application Requirements
Application Composer™	Enable Component Reuse

For more information or
to download free trialware
visit www.webgain.com/goto/jp.html
or call 1-877-WebGain Ext. 15858
Reference code: CMR3

webGAIN™

© 2001 WebGain, Inc. WebGain, the WebGain logo, WebGain Studio, VisualCafe, StructureBuilder, TopLink, WebGain Application Composer, WebGain Quality Analyzer, WebGain Business Designer, and other WebGain product names are registered trademarks and service marks, or trademarks and service marks, of WebGain, Inc. in the U.S. and other countries. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the U.S. and other countries.

Raise the speed limit

Make your code run so fast it will feel illegal

Need to push your Java™ application into the fast lane? Pick up the Optimizeit™ Suite and hold on tight. Three easy-to-use tools fit right into your Java environment. Suddenly you've got a real-time view into the virtual machine. Spot and cure memory leaks and performance bottlenecks. Catch fatal thread issues before they slip into production.

Letting you create fast, reliable, and scalable Java apps is what the Optimizeit Suite is all about. To make sure your code is built for speed just make Optimizeit a regular part of your development process.

Optimizeit Suite™

How fast can you go? Try the Suite and see.

www.optimizeit.com

Make Progress with JSequence

This simple component makes it easy to show users the progress and history of a sequence of events

by Claude DUGUAY

Claude Duguay has developed software for more than 20 years. He is passionate about Java, reads about it incessantly, and writes about it as often as possible. You can reach him at **claude.duguay@verizon.net**.

The progress of a sequence of events often needs to be conveyed to end users in a manner that clarifies what's going on. This requirement is common for installation programs, diagnostic software, automated updates, and protocols, to name just a few. Let's build a simple component that makes it easy to show users the progress and history of a sequence of events (see Figure 1).

As always, we want to maximize flexibility and reusability while keeping the design as simple as possible. There are several ways to approach how we have the items in a sequence report their progress.

Because we want to report progress using a JProgressBar for each item, we could pass a reference to it and leave it up to developers to set the minimum, maximum, and current values as processing continues. With this approach, the sequence of events would be handled outside the JSequence component, which would then act merely as a view to show what was happening. That turns out to be more work than we'd like to impose on developers.

What we really want is a simple mechanism that lets us register sequence items with JSequence and has them report their progress as JSequence triggers each in turn. To do this, we need a notification mechanism, typical of a good model, and a class with default behavior from which we can inherit to simplify the development process. Developers need only to implement an interface to qualify a class as a sequence item. Here's what the interface looks like:

```
public interface SequenceItem
{
  public void
    addSequenceListener(
    SequenceListener listener);
  public void
    removeSequenceListener(
    SequenceListener listener);
  public String getText();
  public void execute();
}
```

Notice that we support adding and removing listeners and allow a text value to be retrieved. The text is used to show users a description of the item. The execute method is called when the item's turn comes up in the sequence. The SequenceListener interface is also simple:

```
public interface SequenceListener
{
  public void progressReady(
    SequenceEvent event);
  public void progressUpdate(
    SequenceEvent event);
  public void progressDone(
    SequenceEvent event);
}
```

JSequence is notified when an item's execution starts with a progressReady() message. As progress continues, several progressUpdate() messages will be received and progressDone() should be called at the end of the item's execution. The execute method should be a nonblocking call, so that it returns immediately, typically starting a thread to do the work it needs to accomplish. It's only when the progressDone() message is received that the next item's execute method will be called.

We won't list the SequenceEvent class (it is available online at www.javapro.com), which merely extends the Swing ChangeEvent class and adds two arguments: one to maintain the state and another to hold an integer value. The states are declared in constants named STATE_READY, STATE_UPDATE, and STATE_DONE, mapping directly to the methods of the SequenceListener interface. The values represent different things, depending on the state.

With STATE_READY, the value must be the maximum value of the progress bar. With STATE_UPDATE, the value is the current relative value. It is assumed that the progress bar's minimum value is zero and that the maximum value was passed when the progressReady() listener method was trig-

gered. The progressUpdate() method typically will be called several times as progress continues. With STATE_DONE, we expect the maximum value again. This serves to confirm that we are, in fact, at the last possible value and to make sure that the associated progress bar is at the maximum value when we're done.

When you look at the class diagram in Figure 2, you'll recognize the SequenceItem interface, SequenceListener, and SequenceEvent classes we've already talked about. SequenceItemList contains a list of SequenceItem objects in the order they will be executed. Three methods are exposed: addSequenceItem(), removeSequenceItem(), and getSequenceItem(), the last of which is used by JSequence to fetch the next SequenceItem when it's ready.

The SequenceItemView takes responsibility for drawing a given item's progress, complete with the text description and state icons (see Listing 1). Let's take a closer look.

The SequenceItemView is effectively a JLabel and JProgressBar component. The code provides instance variables for each, as well as a set of predefined ImageIcon instances to represent blank, ready, update, and done icons. We expect to find these in a subdirectory called images.

The constructor initializes our JLabel with a blank icon and empty string. We put the label in the NORTH part of a BorderLayout and a JProgressBar in the CENTER. For esthetic reasons, we add an empty border for spacing, make the label and progress bar transparent to use a single background color for the view, and use an empty border to place the progress bar slightly indented to the right. This helps align the text and progress bar to the right of the icon.

The last thing the constructor does is make the component invisible. This is important, because we intend to show the views only when it's time for the SequenceItem to execute.

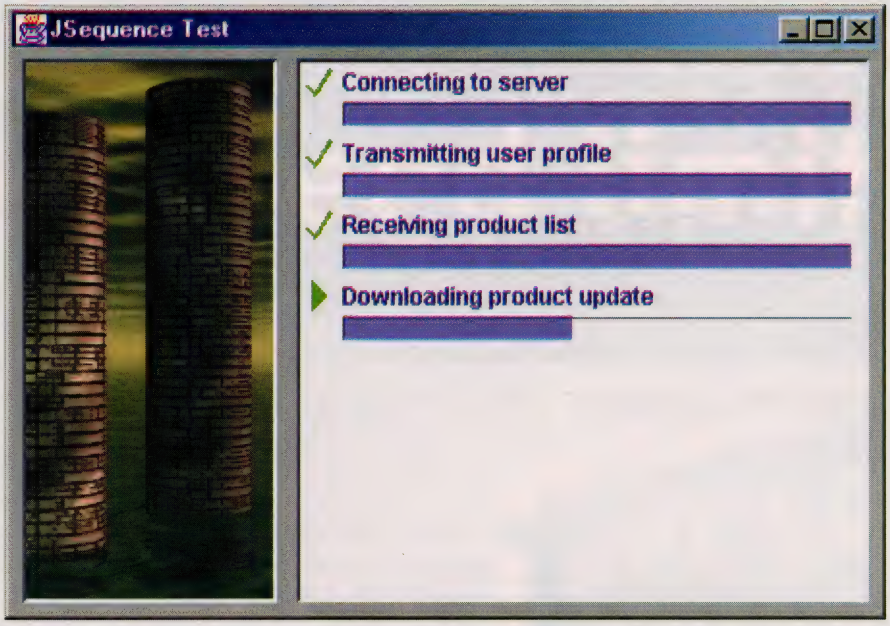


Figure 1 | You are Here JSequence shows the user's progress as the fourth item is processed.

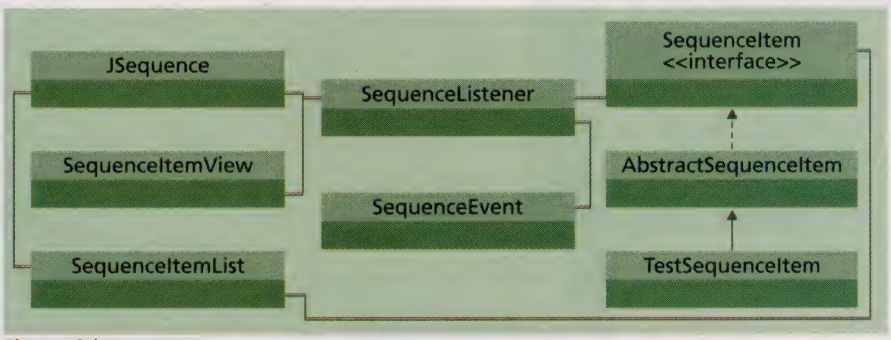


Figure 2 | Divided Classes JSequence classes can be divided into SequenceItem objects, notification infrastructure, and the visual representation classes.

When the item becomes visible in a scroll pane, it may be positioned below the scrolling area, so we implement a scrollToVisible() method that uses the scrollRectToVisible() method in a JViewport. For this to work, however, we have to determine if there is a JViewport in this object's parentage and then make a suitable call. To avoid flickering or other unexpected display problems, we avoid making the call when the rectangle is already visible.

> What we want is a simple mechanism that lets us register sequence items with JSequence and has them report their progress as JSequence triggers each in turn

The next three methods implement the SequenceListener interface. We respond to a progressReady() call by setting the text from the SequenceItem in the JLabel, setting the ready icon, and then initializing the progress bar with suitable minimum, maximum, and current value settings. This is where we make the component visible and call scrollToVisible() to be sure any necessary scrolling takes place.

The progressUpdate() method makes sure the update icon is active and updates the progress bar value with the current value of the SequenceEvent. Just in case the view is not yet visible, we call scrollToVisible() again. Because the method checks to be sure the code is not needlessly executed, this is a small price to pay to ensure proper visibility.

The final method from the SequenceListener interface is progressDone(), which sets the progress bar to its maximum value, sets the icon to done, and waits 100 milliseconds to be sure the interface has a chance

to draw the final state of the view. You'll notice that there is no flow of control to move the sequence forward in the SequenceItemView. That is taken care of in the JSequence component.

Let's take a look at the AbstractSequenceItem class, which provides the basic infrastructure for building a SequenceItem subclass (see Listing 2). Because we use a SequenceItem interface, developers are free to implement these methods directly, but I expect you'll want to keep all that extra work to a minimum.

A SequenceItem has little in the way of state to maintain; only the text needs to be retrieved when the getText() method is called. The constructor reflects this. Most of the work it does is meant to handle SequenceListener registration, removal, and firing the three events supported by a SequenceListener.

> When multiple, sequential processes must be executed, it becomes especially critical that users receive appropriate feedback

We use an ArrayList to manage listeners. The add and remove methods are simple enough and require no explanation. The fireProgressReady(), fireProgressUpdate(), and fireProgressDone() methods are all similar; the only differences between them are the state and value of the SequenceEvent and the listener method called. Each clones the listener list before walking through it and sending the same event to each listener.

Although it's not listed, you'll find a SequenceItemTest class when you download the code for this article. SequenceItemTest uses a Swing timer to step through a phony sequence to test the code and inherits from AbstractSequenceItem.

Continued on page 67.

Listing 1 Progress in View

```
import java.awt.*;
import javax.swing.*;

public class SequenceItemView extends JPanel
  implements SequenceListener
{
  protected JLabel label;
  protected JProgressBar bar;

  protected static final ImageIcon
    BLANK_ICON =
      new ImageIcon("images/blank.gif");
  protected static final ImageIcon
    READY_ICON =
      new ImageIcon("images/ready.gif");
  protected static final ImageIcon
    UPDATE_ICON =
      new ImageIcon("images/update.gif");
  protected static final ImageIcon
    DONE_ICON = new ImageIcon("images/done.gif");

  public SequenceItemView()
  {
    setOpaque(false);
    setBorder(BorderFactory.createEmptyBorder(
      2, 2, 2, 2));
    setLayout(new BorderLayout());
    label = new JLabel(
      "", BLANK_ICON, JLabel.LEFT);
    label.setOpaque(false);
    add(BorderLayout.NORTH, label);

    bar = new JProgressBar();
    bar.setOpaque(false);
    bar.setBorder(
      BorderFactory.createEmptyBorder(
      2, 20, 2, 5));
    add(BorderLayout.CENTER, bar);
    setVisible(false);
  }

  public void scrollToVisible()
  {
    Container ancestor = SwingUtilities.
      getAncestorOfClass(JViewport.class, this);
    if (ancestor != null)
    {
      JViewport viewport = (JViewport)ancestor;
      Rectangle rect = viewport.getViewRect();
      Rectangle bounds = getBounds();
      if (!rect.contains(bounds))
      {
        viewport.scrollRectToVisible(bounds);
      }
    }
  }

  public void progressReady(SequenceEvent event)
  {
    Object source = event.getSource();
    if (source instanceof SequenceItem)
    {
      SequenceItem item = (SequenceItem)source;
      label.setText(item.getText());
    }
    label.setIcon(READY_ICON);
    bar.setMinimum(0);
    bar.setMaximum(event.getValue());
    setVisible(true);
    scrollToVisible();
  }

  public void progressUpdate(SequenceEvent event)
  {
    label.setIcon(UPDATE_ICON);
    bar.setValue(event.getValue());
    scrollToVisible();
  }

  public void progressDone(SequenceEvent event)
  {
    bar.setValue(bar.getMaximum());
    label.setIcon(DONE_ICON);
    sleep(100);
  }

  public static void sleep(int time)
  {
    try
    {
      Thread.sleep(time);
    }
    catch (InterruptedException e) {}
  }
}
```

The SequenceItemView draws a given item's progress, including a text description and state icons.

Listing 2 Basic Abstract

```
import java.util.*;

public abstract class AbstractSequenceItem
  implements SequenceItem
{
  protected String text;
  protected ArrayList listeners = new ArrayList();


  public AbstractSequenceItem(String text)
  {
    this.text = text;
  }

  public String getText()
  {
    return text;
  }

  public void addSequenceListener(
    SequenceListener listener)
  {
    listeners.add(listener);
  }

  public void removeSequenceListener(
    SequenceListener listener)
  {
    listeners.remove(listener);
  }

  public void fireProgressReady(int max)
  {
    ArrayList list =
      (ArrayList)listeners.clone();
    SequenceEvent event = new SequenceEvent(
      this, SequenceEvent.STATE_READY, max);
    for (int i = 0; i < list.size(); i++)
    {
      SequenceListener listener =
        (SequenceListener)list.get(i);
      listener.progressReady(event);
    }
  }

  public void fireProgressUpdate(int val)
  {
    ArrayList list =
      (ArrayList)listeners.clone();
    SequenceEvent event = new SequenceEvent(
      this, SequenceEvent.STATE_UPDATE, val);
    for (int i = 0; i < list.size(); i++)
    {
      SequenceListener listener =
        (SequenceListener)list.get(i);
      listener.progressUpdate(event);
    }
  }

  public void fireProgressDone(int max)
  {
    ArrayList list =
      (ArrayList)listeners.clone();
    SequenceEvent event = new SequenceEvent(
      this, SequenceEvent.STATE_DONE, max);
    for (int i = 0; i < list.size(); i++)
    {
      SequenceListener listener =
        (SequenceListener)list.get(i);
      listener.progressDone(event);
    }
  }
}
```

A SequenceItem is responsible mainly for handling SequenceListener registration and removal, and for firing events supported by SequenceListener.

Listing 3 Sequence of Objects

```
import java.awt.*;
import javax.swing.*;

public class JSequence extends JPanel
  implements SequenceListener
{
  protected SequenceList list;
  protected int current = 0;

  public JSequence(SequenceList list)
  {
    setBackground(Color.white);
    setOpaque(false);
    this.list = list;
    setLayout(new BoxLayout(
      this, BoxLayout.Y_AXIS));
    int count = list.getSequenceItemCount();
    for (int i = 0; i < count; i++)
    {
      SequenceItemView view =
        new SequenceItemView();
      SequenceItem item = list.getSequenceItem(i);
      item.addSequenceListener(this);
      item.addSequenceListener(view);
      add(view);
    }
  }

  public void execute()
  {
    int count = list.getSequenceItemCount();
    if (count > 0)
    {
      current = 0;
      list.getSequenceItem(0).execute();
    }
  }

  public void progressReady(SequenceEvent event) {}
  public void progressUpdate(SequenceEvent event) {}
  public void progressDone(SequenceEvent event)
  {
    int count = list.getSequenceItemCount();
    current++;
    if (current < count)
    {
      list.getSequenceItem(current).execute();
    }
  }

  public JScrollPane getScrollPane()
  {
    JPanel panel = new JPanel(
      new BorderLayout());
    panel.setBackground(getBackground());
    panel.add(BorderLayout.NORTH, this);
    return new JScrollPane(panel);
  }
}
```

In JSequence, the main class for this component, the constructor expects an instance of SequenceList, which is a collection of SequenceItem objects in the order you want to trigger them. From this collection of SequenceItem objects, JSequence will then build a set of SequenceItemView objects and put them in a BoxLayout.

Contined from page 65.

If you take a look at the actionPerformed() method, which is triggered on each timer tick, you'll see how easy it is to fire off SequenceListener events as you process things in your own code.

JSequence is the main class for this component (see Listing 3). The constructor expects an instance of SequenceList, which is a collection of SequenceItem objects in the order you want to trigger them. From the SequenceList, JSequence will build a set of SequenceItemView objects and put them in a BoxLayout. You'll find a JSequenceTest class online that demonstrates JSequence usage.

During the loop that creates the views, we register both the SequenceItemView and the JSequence classes as implementations of the SequenceListener interface, ready to receive notification from SequenceItem objects. The SequenceItemView does most of the work with SequenceEvents, but JSequence watches for the progressDone() message and moves on to the next item when it receives it.

You'll notice a getScrollPane() method, the purpose of which may not seem obvious. If you put JSequence, which uses the BoxLayout, directly in a JScrollPane, you'll find that the view expands to fill the shape of the scroll pane. Because what you really want is to put JSequence at the top and let it expand to fill space going down as new items are executed, you need to put JSequence at the NORTH end of a BorderLayout first. To make this easy, you can call getScrollPane() instead of doing the work separately. To use it, create a JSequence instance and put the JScrollPane returned by getScrollPane() in your application's layout.

JSequence provides a simple infrastructure that allows you to show progress through several sequential processes. JSequence can be given any series of objects that implement the SequenceItem interface. It will step though each item, showing a user's progress along the way. Each item will be shown in a list view, allowing users to look back at each completed item, so they can see what has been done at any point in the process.

When multiple, sequential processes must be executed, it becomes especially critical that users receive appropriate feedback. Without it, their expectations are ill managed and they may look for relief by clicking elsewhere on your interface or typing away without guidance. JSequence can help you avoid these problems, showing you each step along the way. JP

Use these Locator+ codes at www.javapro.com to go directly to these related resources.

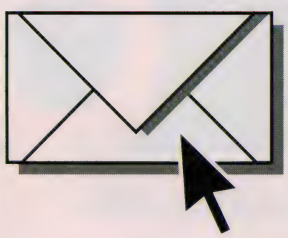
JP0201 Download all of the code for this issue of *Java Pro*.

JP0201VC Download the code for this article.

JP0201VC_T Read this article online.

JAVAinsight

FREE



Weekly E-Mail News from *Java Pro!*

Now you can get the best in Java news, resources, how-to tips and more delivered to your in-box every week. You tell us how much Java you want, and we'll wrap it up and send it to you.

It's fast, it's easy, it's free, and it's chock full of Java.

Sign up online at:

www.javapro.com

And while you're there, check out *Java Pro* online. All the in-depth articles, archives, code, and hands-on information you expect from the publishers of *Java Pro*.

Java is a trademark of Sun Microsystems.

FTP FAWCETTE TECHNICAL PUBLICATIONS

OVERWHELMED BY J2EE COMPLEXITY?
optimalJ™
from Compuware
Compuware can help. With OptimalJ from Compuware, you can generate complete, working applications directly from a visual model. Now you can concentrate on what you're coding—not how.
Try OptimalJ for FREE—log on to www.compuware.com/products/optimalj, and download the OptimalJ Professional Edition to give it a spin today.
• Business Rules Editor
• Pattern-driven Generation
• Active Synchronization
• Integrated Deployment
COMPUWARE
Planning Developing Integrating Testing Implementing Managing Staffing

How to Climb a B-tree

B-tree, a popular tree-indexing structure, is perfect for the db4o database system, and can save you a lot of disk access time

by Rick GREHAN

Some projects go on for longer than we expect. Sometimes they even improve.

One project that developed differently than I expected was an indexing program I worked on. In "Let There be an Index" (Java to Go column, October 2001), I demonstrated how you could extend the capabilities of a lightweight persistent-storage database system by defining a simple linear-array index and storing that index as an object in the database. (The database system was db4o, from www.db4o.com.) At the time, I lamented the fact that you had to read the index from the database. And, because the index was implemented as an array, you had to read the *whole* index from the database.

> A B-tree is like a binary tree on steroids

We could save considerably on disk access time (and memory space) if we had an indexing structure that allowed us to read only part of the index in to memory at a given time and still do all the necessary adding, deleting, and searching that we need an index to do. A popular tree indexing structure fits this bill nicely; it's called the B-tree. Donald Knuth describes the B-tree data structure and its associated algorithms in *The Art of Computer Programming: Sorting and Searching* (Addison-Wesley, 1998). He references the original discoverers of the B-tree; R. Bayer and E. McCreight. (Knuth also mentions that M. Kaufman independently developed the idea of the B-tree at about the same time as did Bayer and McCreight.)

The B-tree is ideal for large indexes in databases because its structure is particularly suitable for external sorting. Knuth says that the routines for searching and updating the B-tree can be done with "comparatively simple algorithms." Of course, this is Knuth talking, and I'm not sure what other algorithms he's comparing B-tree algorithms to. Those with a programming prowess greater than mine may find B-tree algorithms simple. I find them somewhat daunting.

Nevertheless, what I'll provide here is a complete B-tree package in Java, along with a recipe for adding the B-tree routines to a db4o database. The result will be that you'll not only have a library of code that will let you experiment with the powerful B-tree data structure, but you'll be well on your way to creating a persistent version that you can incorporate in your own Java object-oriented databases.

The B-tree

A B-tree is like a binary tree on steroids (see Figure 1), and we can use the idea of the binary tree as the springboard for describing a B-tree. A binary tree consists of a set of linked "nodes." Each node carries a payload—which is usually a key/data pair, though we often simply consider the key portion of the payload—and left and right child pointers. For any given node in the binary tree, the left child pointer points to the subtree containing all keys less than the node's key entry; the right child pointer points to the subtree containing all keys greater than the node's key. (Here, the terms "less than" and "greater than" depend on whatever algorithm we're using to

Rick Grehan is a technical evangelist with Compuware/NuMega. Rick was a senior editor at *BYTE* magazine (where he wrote the JavaTalk column) and at *Computer Design* magazine. He can be reached at rgrehan@top.monad.net.

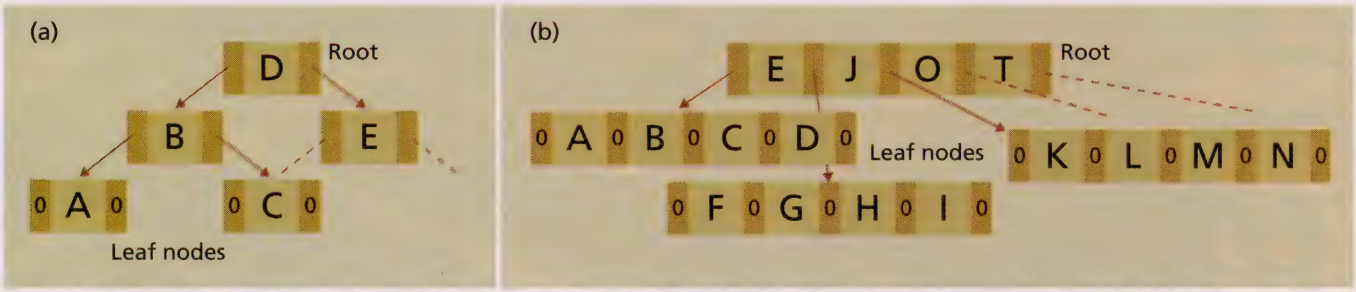


Figure 1 | Pumped-up Binary Tree You can imagine the B-tree as being a beefy binary tree. (a) Each binary tree carries a single payload item (usually a key/data pair—here represented as single letters) and two child pointers. (b) Each B-tree node holds multiple keys and pointers. There is a maximum number of keys that can be on a given B-tree node. If that maximum is given by *m*, then the tree is a *B-tree of order m*. The B-tree in (b) is probably a B-tree of order 4.

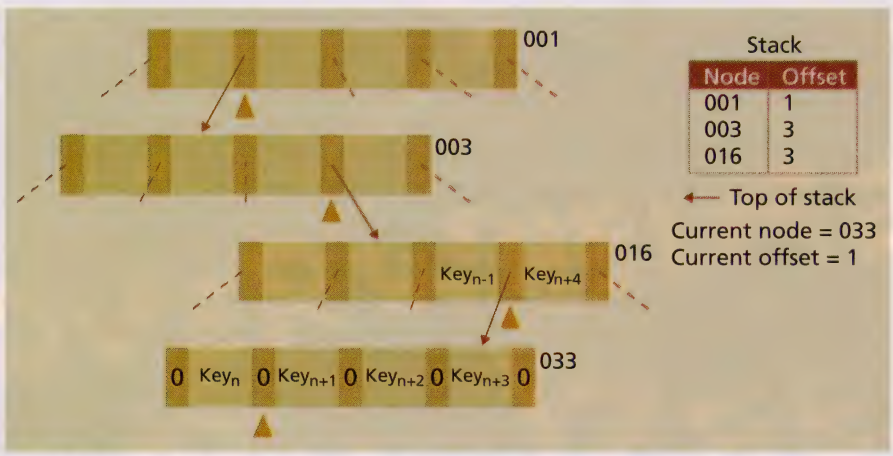


Figure 2 | Search a B-tree As the search algorithm climbs down through a B-tree, it stores a history of its trail on a stack. Starting at the root, the search algorithm pushes the node reference (here, we've given each node a number to make the algorithm clearer) and offset to the child node pointer that's followed to the target key. In this example, the algorithm was searching for Key$_{n+1}$, which has been found on node 033 at offset 1.

order the keys. For strings, it might be lexicographical ordering; for dates, we might want the index ordered in ascending or descending date; and so on.)

The "topmost" node in a binary tree is its "root." The root points to children nodes, which—in turn—may or may not point to more children nodes depending on the height (or depth, if you like to look at the tree upside down) of the tree. Nodes farthest from the root have no children. These terminal nodes are sometimes referred to as "leaf nodes" or "leaves." (I suppose, then, we *are* looking at an upside-down tree. The root is at the top, the leaves are at the bottom.)

We can use this same terminology when we talk about a B-tree. A B-tree node carries a payload and possibly some pointers to other nodes ("possibly" because the pointers on leaf nodes are empty). A B-tree's topmost node is its root; nodes farthest from the root are the leaf nodes; a node can be a parent of multiple nodes—which are its children; and so on.

The difference is that, in a B-tree, each node holds multiple keys. In the strict definition of a B-tree, each node is allowed to hold a specified maximum number of keys. If that maximum number is *n* for a particular B-tree instance, then we would say that instance is a *B-tree of order n*.

Just as in a binary tree, the keys on a node are flanked by pointers to children nodes. For a given key on a given node, the left pointer points to the subtree holding all keys less than that key; the right key pointer points to the subtree holding all keys greater than that key. (Works pretty much the same way a binary tree works, right?) Note that, if a node has *n* keys and is not a leaf node, then that node will have *n+1* children nodes.

Again, though we often simply refer to the key portion of a node's payload, in practical applications, the payload is a key/data pair. It works like an instance of the Java Dictionary class. Keys are what you search for; they are the unique identifiers in the index. Each key is associated with a data item. So, for example, I might use the keys to hold peoples' social security numbers, and the associated data item would be an object holding the person's name, address, phone number, and so on.

That's presuming that the data item can be an arbitrarily complex object. There's no reason why it can't, but in most implementations of a B-tree, the associated data item is a record number or an offset to a location in a file where the data is stored. The idea is to keep the data component in the B-tree small, because a B-tree is meant to be a disk-based index structure. When the system searches a B-tree node, that node has to be loaded from disk (at least once). You don't want unnecessary stuff stored in the node, because each disk read costs you time. Certainly you want the keys stored in the node, but the data item should be simply a pointer. When the B-tree is being searched, the search algorithm is interested only in the keys and node pointers. The data pointer is important only when the target key is finally found.

Searching a B-tree

The efficiency of the B-tree's search algorithm is one of the structure's hallmarks. Once again, we'll use the binary tree as a stepping-stone to our understanding of how a B-tree works. Suppose I wanted to find a key in a binary tree (refer to Figure 1 again). My search algorithm would begin at the root and determine whether the target key is greater than, less than, or equal to the key on the root. If equal, the algorithm is done. If less, the algorithm follows the left child pointer and repeats the search process. If greater, the algorithm follows the right child pointer. The search algo-

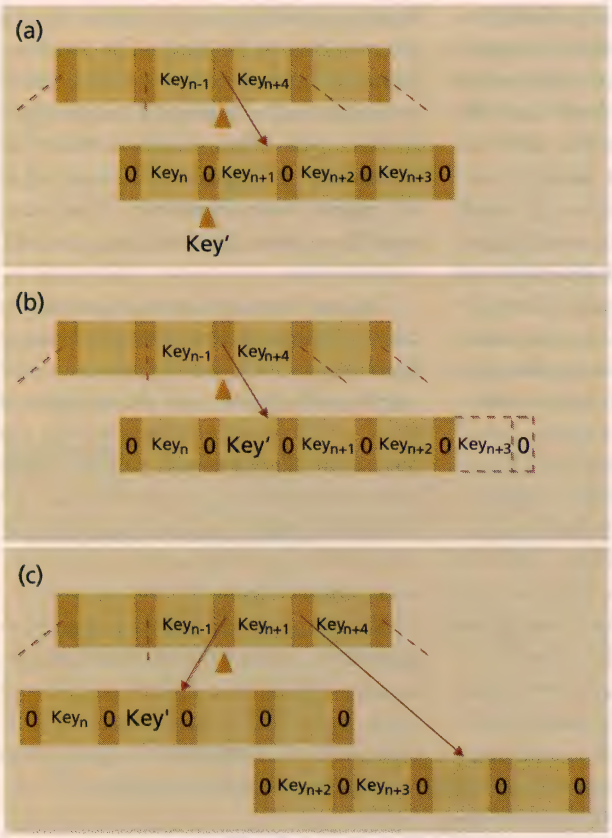


Figure 3 | Insert a Key (a) The insert algorithm has determined that Key' is to be inserted between Key$_n$ and Key$_{n+1}$. (b) The algorithm inserts the key in its proper location on the node, but doing so overflows the node. (Remember, the B-tree allows no more than a specified maximum number of keys on a node.) (c) So, the algorithm splits the node, forming left and right sibling nodes. The middle key of the original node—Key$_{n+1}$—percolates up the tree and is inserted in the parent node. (Note that the insertion in the parent node may cause that node to split, and the process may repeat all the way up to the root.)

rithm keeps going until it either finds the key—success!—or reaches a leaf node. If it hits a leaf node, the key is not in the tree and the algorithm reports the failure.

Notice that each comparison effectively cuts the number of keys that have to be searched in half—assuming a perfectly "balanced" tree. Keep that in mind.

Now consider the search algorithm for a B-tree. As with a binary tree, the search begins at the root. The B-tree search algorithm scans through the keys on the node. If the key is found, the algorithm stops (and returns the data item). If the key is not found, the algorithm "pauses" at the location on the node where the key *would have been* had it been on that node (see Figure 2). If you can imagine the algorithm controlling a little arrow, the arrow ends up between keys. So the arrow is aimed at either (a) a pointer to a child node, or (b) an empty pointer. (It's as if the arrow is saying: "The key is not on *this* node, but—if it's in the tree—it's in the subtree that you can find by following the node pointer I'm aimed at.") If (a), then the algorithm follows the pointer to the child node, and searches *that* node (the same as it did in the root). This process continues (as in the binary tree) until either the key is found or the algorithm hits a leaf node—which is case (b)—and stops.

Now we can see why a B-tree outperforms a binary tree when used for external sorting (for example, when the index is stored out on disk). For a binary tree, when you fetch a node—which will require a slow disk read—the result of scanning that node cuts in half the number of keys that have to be searched. The next node fetch cuts the tree in half again, the next in half again, and so on. In general (ignoring caching), each slow disk fetch cuts the number of keys to search in half. However, with a B-tree, fetching a node and scanning it cuts the number of keys that have to be searched by approximately $1/n$, where n is the number of keys on a node. So, for example, if you have (on average) 10 keys on a node, the first fetch reduces the number of keys that must be searched to 1/10th of the original, the next fetch to 1/10th of that, the next to 1/10th of that, and so on. (Knuth provides detailed mathematics for the upper bounds of the B-tree's performance.)

Traversing the Tree

Traversing the tree means moving through it in some orderly fashion. An algorithm that implements an *inorder* traversal visits the keys in their sorted order (lowest to highest). This would be implemented with a "seek next key" method. We would also want to have a "seek previous key" method, which would provide what amounts to a "reverse-inorder" traversal of the tree (highest to lowest). I'll use the "seek next" and "seek previous" terminologies (instead of inorder and reverse-inorder) because they're easier to remember and fit well with the names of the methods in the code.

To implement algorithms for traversing the tree and make them useful, we need to add the concept of a user's location in the tree (remember the little arrow?). Whenever an operation is performed on the tree—adding, deleting, searching, and so on—the B-tree code must remember where the caller is in the tree. In other words, if I call the "seek next key" method, the B-tree code must know my current key location (where I'm at; where the arrow is pointing) to deduce the next key in the tree.

Our B-tree code could implement location in the tree by simply remembering the target key of the most recent operation, and performing an implicit search operation before carrying out a "seek next key" or "seek previous key" operation. So, if I add key A and then call a seek next key operation, the "seek next key" algorithm would first do a search for key A, then seek to the next key. But that would involve unnecessary reseeking at the beginning of any sequential search operation.

Instead, we add a stack to our B-tree implementation that will also help us when we have to do inserts and deletes (more on this later). You can see how this works by looking at Figure 2. (We've numbered the nodes in this figure to make things clearer. We might imagine that those numbers are records in a file on which each node is stored.) In this figure, the caller has asked the search algorithm to locate Key$_{n+1}$. As the search algorithm moves down the tree, it pushes onto the stack the node number and offset of those nodes it passes through. When the algorithm finishes, it stores the current node number and offset of the located key. The contents of the stack, combined with the current node and current offset, constitute the caller's location in the B-tree.

So, if the user calls the "seek next key" function, the algorithm simply advances the current offset to 2, and returns Key$_{n+2}$. Another call returns Key$_{n+3}$.

What happens when you make yet another call to "seek next key"? The algo-

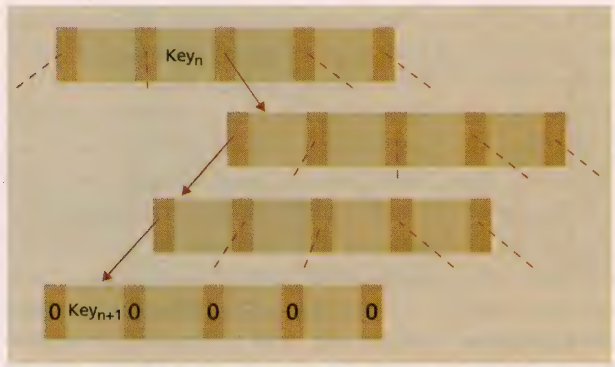


Figure 4 | A Successor Key In this B-tree, the delete algorithm has been asked to delete Key$_n$. But the algorithm cannot simply remove Key$_n$ from the node because it is flanked by two non-null child pointers. Instead, the delete algorithm follows the right child pointer to the lowest leftmost key on that subtree—Key$_{n+1}$, the successor key of Key$_n$. The algorithm then overwrites Key$_n$ with Key$_{n+1}$, then deletes Key$_{n+1}$ from the leaf node. (Key$_{n+1}$ on the leaf node is flanked by null child pointers.)

rithm advances the offset and recognizes that it has "fallen off" the right-hand side of the node. It then pops the stack, sets the current node to 003, sets the current offset to 3, and returns Key$_{n+4}$.

It's fairly easy to see how the principles behind "seek next key" could be applied to build a "seek previous key" algorithm, so I'll skip the discussion of that. But I will mention "rewind" and "toEnd" operations, which set the caller's location in the tree to before the first key or after the last key, respectively. Our B-tree will need to implement these so we can step through the whole tree—forward or backward. They're pretty easy to concoct. The rewind operation simply starts at the root, follows the leftmost child node pointers until it encounters a leaf node, and sets the current location to the far left of the node (so that a "seek next key" operation returns the lowest, leftmost key—the "first" key in the tree). Similarly, the toEnd operation starts at the root, follows the *rightmost* child node pointers to a leaf node, and sets the location to the far right of the node (so that a "seek previous key" operation returns the lowest, rightmost key in the tree).

Decorating the B-tree

If a B-tree is to be of any use to us, we have to be able to put things in it. We must have an insert algorithm so we can put keys (and associated data items) into the tree. And, of course, the keys have to be placed in the B-tree in their proper places.

The insert algorithm must begin with a search algorithm. This serves two purposes. First, the search verifies that that key is not already in the B-tree. (How the insert algorithm proceeds from there depends on how we implement it. We might decide to return an error code indicating that a key has already been found—which is what the code I provide will do. Or, the insert algorithm might—without indicating an error—overwrite the old data item with the new one.) Second, doing the search sets up the stack (see Figure 2), building a kind of trail from the root to the location where the key will be added. You'll see why we need this trail in a moment.

Note that insertion always occurs at a leaf. The search algorithm terminates (assuming the key is not found) when it hits a leaf. The current page and offset indicate where the new key should be added (remember, the search algorithm leaves the "arrow" pointing at where the key *would* be if it were on the node).

The insert algorithm is best explained with an illustration (see Figure 3). Suppose we have a key—call it Key'—that we want to insert in a B-tree. We call the insert method, which calls the search method. Search discovers that Key' is not in the tree and, furthermore, that it should be inserted between Key$_n$ and Key$_{n+1}$ (Figure 3a). The insert algorithm "opens" a slot on the node, moving Key$_{n+1}$, Key$_{n+2}$, and Key$_{n+3}$ to the right, and inserting Key' into that slot. Now, if the maximum number of keys on the node has not been exceeded, the algorithm completes and the new key—Key'—is safely in the tree.

However, if the insertion does exceed the maximum number of keys allowed on a node (Figure 3b), then the insert algorithm must split the node. The algorithm locates the center key on the node, Key$_{n+1}$ in this case. The keys to the left of Key$_{n+1}$ are left on the child node; the keys to the right of Key$_{n+1}$ are taken off of the child node and placed on a new node. The algorithm then looks at the top stack entry (remember, the stack contents were built up as search moved through the tree looking for Key'). The top entry points to the parent node and the offset of the pointer to the child node that is about to be split. The offset points to where the center key—Key$_{n+1}$—must be inserted in the parent node (Figure 3c). The insert algorithm inserts the center key into the parent node.

So, inserting Key' has caused a node to divide, and the center key of the divided node "bubbles" up the tree to be inserted in the parent node. Notice that the insertion of Key$_{n+1}$ of the parent node might cause *that* node to overflow and split, and the whole process can continue up the tree until the root is reached (and if the root is split, then the center of that node is placed on a brand new node, which becomes the tree's new root).

We can now see why the stack is so important. It maps the trail that the split operations must follow up the tree, should splits occur when a key is inserted. The stack plays an important role again, when we examine the delete algorithm.

Pruning the Tree

It's not too far off to say that the delete operation is the reverse of the insert operation—in implementation as well as effect. The delete algorithm begins with a search operation, which again serves two purposes: to verify that the key to be deleted is in the tree (obviously an error is returned if it is not), and to build a trail through the tree on the stack.

If the key to delete is on a leaf, the process is trivial. The key is simply removed from the node by shuffling all of the keys (and associated data items) one slot to the left. We could leave it at that, but the B-tree might end up imbalanced. That is, if lots of keys are deleted from the node, it ends up mostly empty and our B-tree loses some of its efficiency.

Instead, what the delete algorithm should do, when possible, is the reverse of the insert operation's split (I call this reverse split a "coalesce"). The delete algorithm examines the node's sibling nodes and counts the number of keys on those siblings. If the number of keys on the current node, plus the number of keys on one of its siblings, plus one, is less than the maximum number of keys allowed on a node, then the current node and the sibling can be combined into

a single node—coalesced. The parent key is pulled from the parent node and inserted in the midst of the coalesced nodes. (Return to Figure 3, and imagine that the tree is in the state as shown in Figure 3c. Key' is deleted from the tree. The coalescing process will combine the two leaf nodes in Figure 3c, and the tree will end up in the state as shown in Figure 3a.)

Of course, because a key is pulled off a parent node, that node shrinks by one key. The coalescing process might percolate up the tree (just as the split operation did).

If the key to delete is not on a leaf node, the process is trickier. We can't just pull the key off the node: What would we do with the two flanking node pointers? (They can't be combined. We can't just bash the two subtrees together and hope for the best.) The answer is that we get a successor or predecessor key with which to overwrite the key that's being deleted.

Look at Figure 4. Suppose we want to delete Key_n. The delete algorithm works by locating the inorder successor of Key_n, which in this tree is Key_{n+1} (the lowest, leftmost key on Key_n's right subtree). It then copies Key_{n+1} (and its associated data item) over Key_n, replacing the latter with the former. The algorithm then deletes the copy of Key_{n+1} that's on the leaf. Because the delete is taking place on a leaf node, the process can continue as described. (Note that we could have used Key_n's inorder predecessor—Key_{n-1}—just as easily. Either would work. In fact, a good delete algorithm would balance using the two in an effort to keep the tree from becoming lopsided.)

Listing 1 A bTreeKeyPage

```
// Class definition for a key page in the bTree
public class bTreeKeyPage
{
  public static int MAXKEYS = 4;
  public int numKeys;
  public bTreeKey[] keyArray;
  public bTreeKeyPage[] pageArray;
  public Object[] dataArray;

  // Constructor
  public bTreeKeyPage()
  {
    numKeys=0;  // No keys on page yet

    // Pre-allocate the arrays -
    // but for now they're all NULL
    keyArray = new bTreeKey[MAXKEYS];
    pageArray = new bTreeKeyPage[MAXKEYS+1];
    dataArray = new Object[MAXKEYS];
  }

  // search
  // Given a btreekey object, search
  // for that object on this page.
  // Returns loc.
  // If loc>=0 then the key was found
  // on this page and loc is index of
  // located key. If loc<0 then the key
  // was not found on this page and
  // abs(loc)-1 is index of where the
  // key SHOULD be.
  public int search(bTreeKey targetKey)
  {
    int lo = 0;
    int hi = this.numKeys-1;
    int mid = 0;

    if(keyArray[lo].compare(targetKey)>0)
      return(-1);
    do {
      mid = (lo+hi)/2;
      if(keyArray[mid].compare(targetKey)
        ==0)
        return(mid);
      if(keyArray[mid].compare(targetKey)
        >0)
        hi = mid - 1;
      else
        lo = mid + 1;
    } while(lo <= hi);
    return(-lo-1);
  }

  // insert
  // Insert the btreekey object  and data
  // object at index. Everything on page
  // is slid "to the right" to make space.
  // numKeys is incremented
  public void insert(bTreeKey newKey,
    Object newData, int index)
  {
    // If adding to right, no moving to do
    if(index<numKeys)
      // Shuffle up
      for(int i=numKeys-1;i>=index;i--)
      { keyArray[i+1]=keyArray[i];
        pageArray[i+2]=pageArray[i+1];
        dataArray[i+1]=dataArray[i];
      }

    // Insert new items
    keyArray[index]=newKey;
    dataArray[index]=newData;

    numKeys++;
  }

  // delete
  // Delete the btreekey reference,
  // keypage reference, and data
  // reference at index. Everything
  // on page is slid "to the left"
  // numKeys is decremented
  public void delete(int index)
  {
    // If rightmost key..nothing to do
    if(index<numKeys-1)
      // Shuffle down
      for(int i=index;i<numKeys;i++)
      {
        keyArray[i]=keyArray[i+1];
        pageArray[i+1]=pageArray[i+2];
        dataArray[i]=dataArray[i+1];
      }

      // Decrement key count and null
      // rightmost item on node
      numKeys--;
      keyArray[numKeys]=null;
      pageArray[numKeys+1]=null;
      dataArray[numKeys]=null;
    }
  }
```

This source code for a key page in the B-tree demonstrate routines that allow for adding key/objects, deleting, searching, and traversing the tree in both directions.

A Java B-tree

Now that we know the routines for adding, removing, searching, and traversing a B-tree, we're ready to turn it into Java code.

Listing 1 shows the Java code for the bTreeKeyPage class. This class implements what amounts to B-tree nodes. (I use the term "page" in the source code instead of "node." This is a holdover from the fact that early implementations of B-trees stored each node on a disk page.) The data portion of the class includes an integer that tracks the number of keys on the page. It also includes a static integer that defines the maximum number of keys on a page. I've set it to 4 to correspond to the illustrations I've shown, but you can set it to whatever you'd like.

The bTreeKeyPage object has three arrays:

- keyArray[]—This array holds the references to the key objects.
- pageArray[]—This array holds the references to children nodes.
- dataArray[]—This array holds the references to data objects.

Each keyArray[] entry has a corresponding dataArray[] entry. There will be one more pageArray[] entry than keyArray[] entries (because of the rightmost child pointer on the node).

A bTreeKeyPage object knows how to do three things: look for a key on itself, insert a key/data pair on itself, and delete a key/data pair from itself.

Listing 2 Key Object Interface Definition

```
// Interface for bTreeKeys
public interface bTreeKey
{
  // Compares 'this' with thatkey
  // Returns
  // < 0 if this < thatkey
  // =0 if this = thatkey
  //  > 0 if this > thatkey
  public int compare(bTreeKey thatkey);

  // Return a reference to the key object
  public Object getKey();
}
```

Keys in the bTree must implement this interface. This allows the bTree class methods for searching and inserting to order the keys properly in the index.

Listing 3 String Key

```
// Class for implementing a key
// that's a String
public class bTreeStringKey implements bTreeKey
{
  public String key;  // The key element

  // Comparison method
  public int compare (bTreeKey thatkey)
  {
    return(this.key.compareTo(
      (String)(thatkey.getKey())));
  }

  // Return reference to key object
  public Object getKey()
  { return(this.key);
  }
}
```

This class implements the bTreeKey interface and defines a key that is a simple String datatype.

Listing 4 Testing the Tree

```
public class testbTree
{
  public static void main (String[] args)
  {
    int i;
    bTreeStringKey myKey;
    String myData;
    bTree myBtree;;

    // Create a bTree object
    myBtree = new bTree("Test");

    // Add three keys
    myKey = new bTreeStringKey();
    myKey.key = "Rick";
    myData = new String("Judy");
    myBtree.add(myKey, myData);

    myKey = new bTreeStringKey();
    myKey.key = "Gene";
    myData = new String("Sandi");
    myBtree.add(myKey, myData);

    myKey = new bTreeStringKey();
    myKey.key = "Phil";
    myData = new String("Kate");
    myBtree.add(myKey, myData);

    // Following returns "Sandi"
    myKey = new bTreeStringKey();
    myKey.key = "Gene";
    myData = (String) myBtree.seek(myKey);
    System.out.println(myData);

    // Step through the tree
    myBtree.rewind();
    while(myBtree.gotoNextKey() == 0)
      System.out.println(myBtree.currentObject);

    // Delete "Phil"
    myKey = new bTreeStringKey();
    myKey.key = "Phil";
    myBtree.delete(myKey);

    // Step through the tree backward
    // "Kate" will be gone
    myBtree.toEnd();
    while(myBtree.gotoPrevKey()==0)
      System.out.println(myBtree.currentObject);
  }
}
```

This simple program demonstrates how to add items to the bTree, as well as how to delete items, search, and traverse the tree.

The search() method uses a binary search on the node, so it's reasonably fast—particularly given that nodes are unlikely to have more than 20 or 30 keys. (I suppose you could define a bTreeKeyPage object that allows, say, 500 keys. But you probably wouldn't want to do that unless you're working with insanely large indexes.) Note that search() returns an integer that is positive or negative, depending on whether or not the key is found. If the key is found, then the returned value is positive and indicates the key's offset on the node. If the key is not found, then the returned value is negative and the key would have been on the node at location abs(return value) -1. This allows the return value to serve as that "arrow" that we described earlier.

The insert() and delete() methods are straightforward and simply shuffle key/data items up and down in the arrays.

The source for the bTree class, which implements a B-tree, is available online at www.javapro.com. The important methods in bTree.java are:

- seek()—This method takes as an argument a target key and searches for that key in the B-tree. If the key is found, seek() returns the data item's object reference. If the key isn't found, seek() returns null.
- add()—This method adds a key/data object pair to the B-tree. It returns an integer value that is zero if the operation was successful, or nonzero otherwise. (The nonzero return value will most likely be the static ALREADYEXISTS value, indicating that the key is already in the B-tree.)
- delete()—The delete() method removes the specified key (and its associated data object) from the B-tree. This method also returns an integer value that is zero if the operation is successful, or nonzero otherwise. (The most likely nonzero value is NOTFOUND, indicating that the key to delete is not in the B-tree.)
- gotoNextKey() and gotoPrevKey()—These methods return the current location's successor and predecessor key, respectively. The seek() method sets the current location, even if the key isn't found. The insert() method also maintains the correct location, as does delete(). (So, if my B-tree includes the keys "AA," "BB," and "CC," and I delete "BB," a subsequent call to gotoNextKey() will return "CC.")

These methods return an integer value. If the value is zero, the operation completed successfully, and the key/data object references are in the B-tree variables currentKey and currentObject. A nonzero return value will indicate that the traversal has hit the beginning or end of the index.

- rewind() and toEnd()—These methods implement the algorithms of the same name; they set the current location to the beginning of the index and the end, respectively. Calling rewind() followed by gotoNextKey() returns the first key in the B-tree; calling toEnd() followed by gotoPrevKey() returns the last key in the B-tree.

The code in bTree implements all the functionality described earlier. The add() method handles node splitting, and the delete() method performs coalescing. (In fact, the code for performing a coalesce operation was drawn out into a separate, private method to keep the code more understandable.)

Because the code was written in Java, the node pointers (stored in the array pageArray[]) are object references, as are the references to keys and data objects on the node. This means that we have to be careful about object "loiterers."

For example, when the delete() method removes a key/data pair from a node, the node shrinks. Key/data pairs and child node pointers on the right side of the node are copied to the left to fill in the hole left by the deleted key/data item. If we were writing this in the C language (or some other non-object-oriented language), we wouldn't have to worry. But with Java, we have to null the old rightmost key and data references explicitly. If we don't, the node will be holding either duplicate references (at best) or loitering references (at worst).

Key objects that go into the bTree class must implement the bTreeKey interface (see Listing 2). The only important method in this simple interface is compare(), which enables the bTree algorithms to sort keys as they are inserted and searched for in the index. An example of a simple class implementing the bTreeKey interface is shown in Listing 3. The class is bTreeStringKey, and implements a simple String key class.

The code in Listing 4 shows how to make calls into the bTree library. This program adds three keys to the tree and demonstrates how to fetch, delete, and step sequentially (forward and backward) through the tree.

Improvements and Variations

What I've shown is certainly not the last word in B-tree algorithms. The insertion algorithm I've provided simply splits nodes when adding a key causes a node to become overfull. The result is that nodes tend to be half full. Packing them more densely will make for quicker search times, and this can be accomplished with a process called "rotation."

The idea is simple. When a key is added to a node such that the node exceeds its

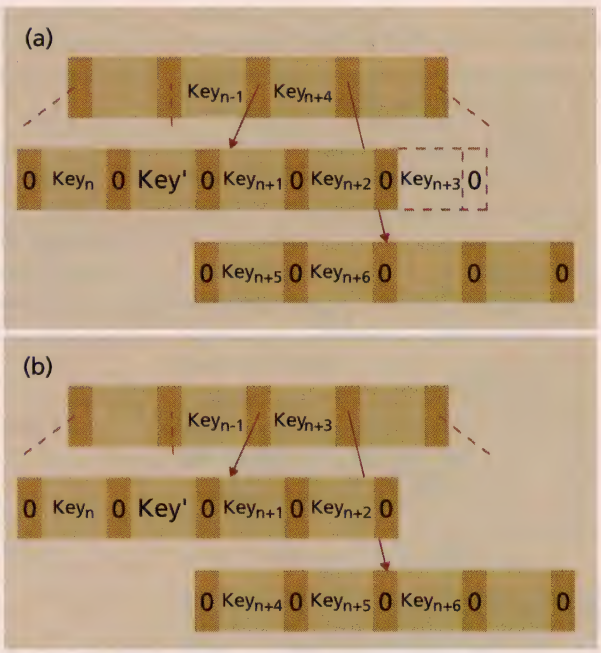

(a)
Key$_{n-1}$
Key$_{n+4}$
0 Key$_n$ 0 Key' 0 Key$_{n+1}$ 0 Key$_{n+2}$ 0 Key$_{n+3}$ 0
0 Key$_{n+5}$ 0 Key$_{n+6}$ 0 0 0
(b)
Key$_{n-1}$
Key$_{n+3}$
0 Key$_n$ 0 Key' 0 Key$_{n+1}$ 0 Key$_{n+2}$ 0
0 Key$_{n+4}$ 0 Key$_{n+5}$ 0 Key$_{n+6}$ 0 0

Figure 5 | Key Rotation B-tree performance can be improved if the insert algorithm resists splitting a node until it's absolutely necessary. (a) In this example, Key' was inserted on a node, causing the node to overflow. Rather than split the node (as in Figure 3), the insert algorithm examines sibling nodes for vacant space. In this case, room was found on the right sibling. (b) The overflow key was moved up to take the place of the parent key, and the parent was moved down into the sibling node.

maximum, rather than split the node, see if one or more of the node's keys can be rotated over to a sibling node. This is illustrated in Figure 5. As you can see, Key' has been inserted in a leaf node, and the node has overflowed. Instead of splitting the leaf into two nodes, the algorithm examines the sibling nodes and discovers room on the right sibling node. As shown in the figure, the keys of the sibling node shift right, the parent key moves down to become the leftmost key on the sibling, and the rightmost key of the overflowed node moves up to take the parent's place. Nodes split only when sibling nodes are full; otherwise, the overflow keys are "poured" into adjacent brother and sister nodes.

Of course, you don't get something for nothing (you never do). Adding code to perform rotation complicates the insert algorithm. Nevertheless, the reward of that complication is that the tree's nodes split less often, so the tree is less tall, packed more densely, and consequently, searches are faster.

In addition, there are variants to the B-tree. Perhaps the most popular is the B+tree. A B+tree keeps all the "active" key/data pairs at the leaves. The interior nodes carry only keys and pointers to children nodes. The nice thing about a B+tree is that traversing the tree is super fast. If you provide pointers to sibling nodes on the leaves, the leaf nodes of the tree are nothing more than a long, doubly linked list.

Random searching results in a slight performance hit in a B+tree, because the search must always go to the leaves. But it's worth it if the application does sequential traversing of the index.

Persistent B-Tree

Of course, the whole point was to create an index that could be stored in a file. We need to make our B-tree persistent. And, with the help of the db4o persistent database (available from www.db4o.com), we can do that. As I described in "OO Databases Get Small Minded" (Java to Go column, September 2001), db4o is a small-footprint object-oriented database written entirely in Java. It uses query-by-example as its query mechanism. This is, to say the least, unusual. However, the database provides persistent storage, it "understands" object references (if object A references object B and you store object A, db4o will automatically store object B), and—to a degree—it provides reachability. With a little work, then, we can put our B-tree into a db4o database and—voila—instant index.

Listing 6 Persistent B-tree Exerciser

```
// db4obTreeTest
import com.db4o.*;
import com.db4o.config.*;
import java.io.File;
import java.lang.System;

public class db4obTreeTest {

  static String skey;
  static String data;
  static bTreeStringKey key;
  static bTree mybTree;
  static ObjectContainer myDB;

  public db4obTreeTest() {
  }

  // Main entry point
  static public void main(String[] args) {
    new db4obTreeTest();

  // Create
  if(args[0].compareToIgnoreCase("-create")==0)
  {
    // Create a new file and index
    new File(args[1]).delete();
    myDB = Db4o.openFile(args[1]);

    // Create btree
    mybTree=new bTree("index1");

    // Store it
    closeindex();

    // Close the container
    myDB.close();
    System.exit(0);
  }

  // Add
  if(args[0].compareToIgnoreCase("-add")==0)
  {
    // Set the activation depth for the file
    Configuration dbConfig = Db4o.configure();
    dbConfig.activationDepth(1);

    // Open the file
    myDB = Db4o.openFile(args[1]);

    // Fetch the index
    fetchindex();

    // Add key and object
    skey = args[2];
    key = new bTreeStringKey();
    key.key = skey;
    data = args[3];
    mybTree.add(key, data);

    // Close the index and file
    closeindex();
    myDB.close();
    System.exit(0);
  }

  // Get
  if(args[0].compareToIgnoreCase("-get")==0)
  {
    // Set the activation depth for the file
    Configuration dbConfig = Db4o.configure();
    dbConfig.activationDepth(1);

    // Open the file
    myDB = Db4o.openFile(args[1]);

    // Fetch the index
    fetchindex();

    // Fetch the object
    key = new bTreeStringKey();
```

Continued on page 77.

This is a simple test program that accepts command-line arguments for manipulating the persistent B-tree. You can create a B-tree, add key/object pairs, and search for keys.

Continued from page 76.

```
    key.key = args[2];
    System.out.println(mybTree.seek(key));

    myDB.close();
    System.exit(0);
  }
  }

  static void fetchindex()
  {
    // Locate the btree
    bTree tempbTree = new bTree("index1");
    tempbTree.keyPageStack = null;
    tempbTree.indexStack = null;
    ObjectSet osbTree = myDB.get(tempbTree);
    if(osbTree.isEmpty())
    { System.out.println("**Index not found");
      myDB.close();
      System.exit(0);
    }
    // Fetch the Btree
    mybTree = (bTree)osbTree.next();
    // Allocate stacks
    mybTree.keyPageStack =
      new bTreeKeyPage[bTree.MAXSTACK];
    mybTree.indexStack = new int[bTree.MAXSTACK];
    // Set the reference to the ObjectContainer
    mybTree.db = myDB;
  }

  // Call before closing file
  static void closeindex()
  {
    // Clear bTree fields out to get
    // rid of loiterers
    mybTree.db=null;
    mybTree.currentPage=null;
    mybTree.currentKey=null;
    mybTree.currentObject=null;
    mybTree.keyPageStack=null;
    mybTree.indexStack=null;

    // Set (update) the btree
    myDB.set(mybTree);
    myDB.commit();
  }

}
```

Listing 5 (available online) provides the code that shows how to make the seek() and add() routines persistent. (The other classes that bTree depends on can remain unchanged. However, for my tests, I removed the delete() method from bTreeKeyPage. I leave it to the reader to make the other routines persistent, such as deleting and sequential searching.)

Listing 6 shows the Java source for opening and closing the database (and reading the index), as well as for adding key/data pairs to the index and reading them out. (This code is based on the October "Let There be an Index" Java to Go column, where I showed how to store a simple array index.)

Note that we don't want to store and fetch the entire B-tree index; that would short-circuit the B-tree's performance (it would perform no better—and probably worse—than the linear array I showed last time). When performing a search operation, we want the search algorithm to fetch only the node that's being searched. And, only that data object associated with the target key is fetched from the database.

You can see this if you examine the readPage() method in Listing 5 (online). This method is called whenever the search routine wants to get a page (a node) from the database. The page is activated, but because the activation depth is set to 1 (the db4obTreeTest class' methods set the activation depth just before opening the database), the contents of the keyArray must be activated explicitly. I activate keys to a depth of 5, which should be sufficient for even the most complex key objects. Consequently, when a node is read from the disk, only the objects immediately needed for searching are loaded.

The simple test program in Listing 6 responds to three commands that you enter as program arguments. You create a file with a B-tree index by entering as the command line argument:

```
-create <filename>
```

This builds the file <filename> and loads into it an empty index named "index1." Once the file has been created, you can start loading keys into the index by entering:

```
-add <filename> <keystring>
  <datastring>
```

So, the command line:

```
-add btree.db Rick Judy
```

will add the key string "Rick" to the file "btree.db" and associate the data string "Judy" with it (assuming "btree.db" was already created with the -create command).

You can fetch items from the database with:

```
-get <filename> <keystring>
```

So, entering:

```
-get btree.db Rick
```

will return "Judy".

This code leaves plenty of room for improvement, but I hope I've provided enough raw material for you to work with. I've shown only how to make the insert and search routines persistent. A little more work would produce a complete B-tree index in a db4o database. And, because the B-tree is named, there's no reason why a single db4o database couldn't hold multiple indexes.

Modularizing the source code would be worthwhile, as well. I can imagine two versions of the persistent B-tree code. One version provides all the capabilities of a full B-tree package—search, add, delete, traverse, and so on. The other, smaller version provides only search capabilities. You could use the "big" version on your desktop and the smaller version on a mobile device. Thanks to db4o's modest size, you can take your index anywhere. JP

GO ONLINE
www.javapro.com

Use these Locator+ codes at www.javapro.com to go directly to these related resources.

JP0201 Download all of the code for this issue of *Java Pro*.

JP0201RG Download the code for this article separately.

JP0201RG_T Read this article online.

Cure Your Waba Woes with a Serial Socket

The search for cheap Palm Internet connectivity can send you down a messy but fruitful path

Some time ago, a reader sent me an e-mail requesting a particular "how to" that I put on the back burner because I was fighting problems on a different front (my apologies to the e-mail's author). Now I've come back to the request, because the question has been nagging me on low volume for several weeks.

The question was about using Java—actually, Waba—on the Palm and took the form: "Dear Rick: How do you do THIS?" I tossed off the question because my initial thought—unnourished by any experience—was that THIS was an easy thing to do, amounting to not much more than plugging the Palm into its cradle and executing a Waba application.

The Deal

So what was "THIS"? Why had the e-mailer requested help?

The Waba SDK comes with several example applications. One is the "Web" application, and it illustrates how to use Waba's Socket class to read a Web page and display its contents on the Palm. The author of the e-mail wrote: "I can get the Web application to run on the desktop (using Waba's bridge classes), but how do I get it to run on the Palm?"

Now, I knew Waba was dependable, so I figured that any problems the reader was having were not due to some failure or lack in the Waba libraries. Consequently, my initial thought was: Just download the Web application to the Palm, put the Palm in its cradle, and launch the application. (I was assuming that the reader—like me—has a Palm that uses the serial-port cradle for Hotsyncing with the desktop.)

After a few weeks of feeling nagged by this question, I realized that I had made an error. Simply running the Web application with the Palm in the cradle doesn't work. We need to create a way for the Socket class to work—that is, to establish a connection with a URL on the Internet—while the Palm is in its cradle. But, when in the cradle, the Palm is simply sitting at the end of a serial connection. Waba's Socket class expects to be able to establish a TCP/IP connection to the URL you pass as an argument to a Socket object's constructor. A serial connection—unaided—is not a TCP/IP connection.

(You might have a modem connected to the Palm's serial port, rather than the cradle. Though I haven't tried it, I'm confident that the Web example would dial out and work fine with a modem. I haven't tried it because I don't have a modem connected to my Palm. I have a T1 Internet connection on my laptop, to which my Palm is connected.)

The solution? Well, I ended up with two solutions to the problem. The first is the "messy" solution. It's messy because I wrote it myself. It's cheaper than the other solution, but its main advantage is that, with a little work, it could be extended to platforms. I call it a Serial Port Socket—sort of a poor-man's implementation of a point-to-point protocol.

The Messy Way

The architecture of my Serial Port Socket system uses what amounts to a client/server structure (see Figure 1). The client side depends on a class called spSocket

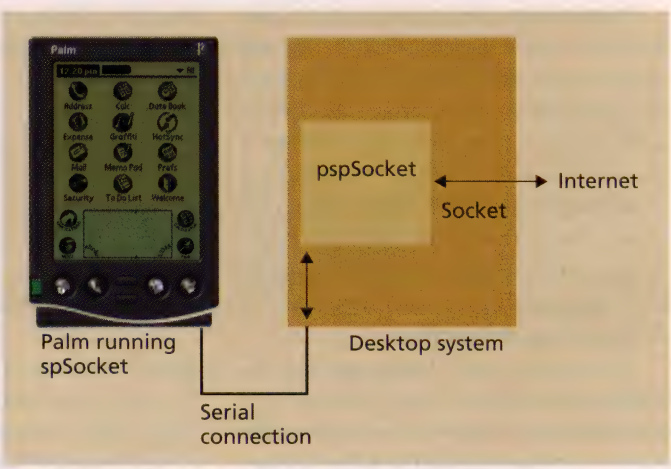


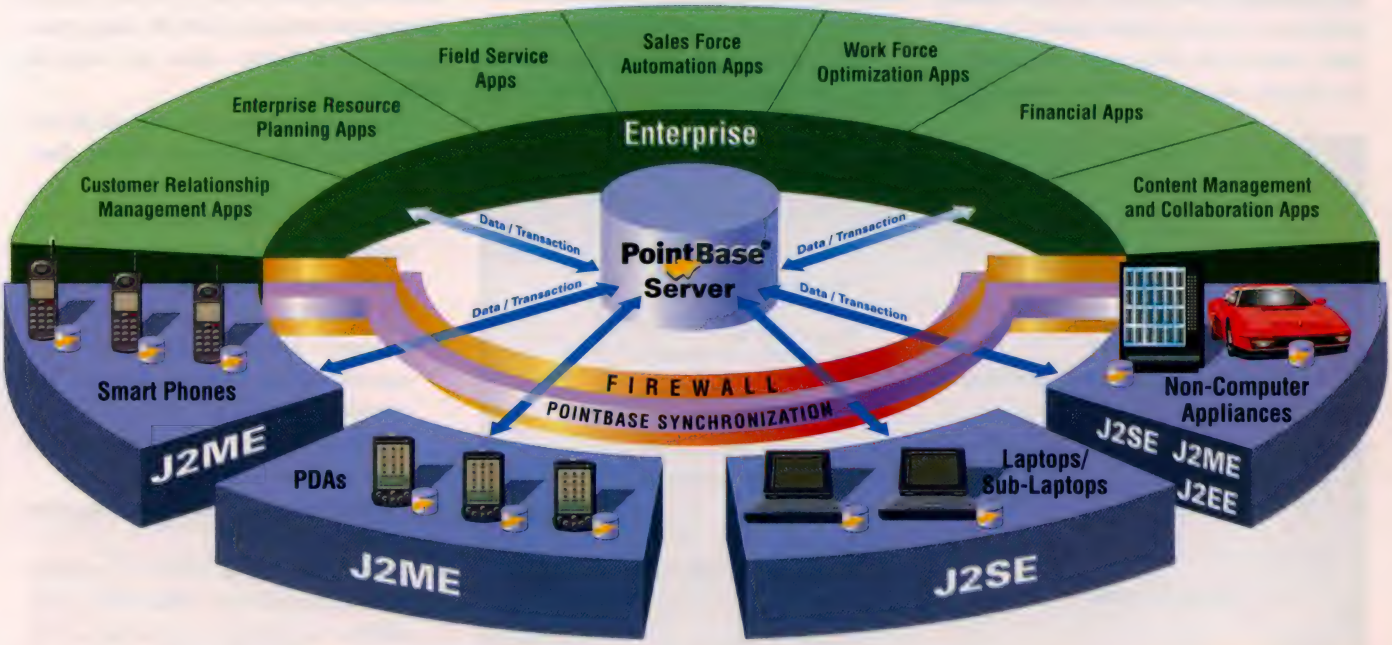
Figure 1 | Architecture of Serial Port Socket The application uses a client/server architecture. The Palm handheld—running a Waba application that uses the spSocket class—communicates with a proxy running on the desktop. The proxy also executes Waba and provides a kind of gateway for the Palm device onto the Internet.

by Rick GREHAN

Rick Grehan is a technical evangelist with Compuware/NuMega. Rick was a senior editor at *BYTE* magazine (where he wrote the JavaTalk column) and at *Computer Design* magazine. He can be reached at rgrehan@top.monad.net.

PointBase®

Synchronization + Relational Database = Enterprise Mobility



PointBase...the Synchronization and Database Solution for Pervasive Computing

PointBase delivers the synchronization and database technology your remote or mobile applications need to store, manage and share information with each other. Designed with the Java™ Platform, SQL, JDBC™ and other industry standards, PointBase provides Java applications with persistent local information storage and universal connectivity via synchronization to other applications and enterprise class databases.

If you are building applications for mobile deployment, then synchronization and persistent storage are a must. The PointBase synchronization and database technology together provide the cornerstone of mobile infrastructure. Features of the PointBase solution allow you to quickly and easily integrate a full function database into your application for on the go persistent storage. PointBase synchronization technology provides bi-directional synchronization between any enterprise database and the PointBase databases providing the mobility and connectivity to the enterprise your applications need.

PointBase Synchronization API
- Bi-directional synchronization between any PointBase database and other enterprise class databases (Oracle, IBM DB2, Microsoft SQL Server, etc.)
- Hub-and-spoke topology
- TCP/IP and HTTP transport protocols
- Failure recovery support ensuring transactional integrity
- Conflict detection and resolution

PointBase Micro
- Relational Database Management System for J2ME™ and PJAVA™
- Ultra small memory footprint (<41K)
- Single connection, thread safe architecture
- Ideal for PDAs and Smart devices

PointBase Embedded
- Relational Database Management System for J2SE™ and J2EE™
- Small memory footprint (approximately 1MB)
- Multiple connections, multi-threaded architecture within the same JVM
- Ideal for integrated Java applications operating on laptops, sub laptops and pervasive computing devices and non-computer appliances

PointBase Server
- Relational Database Management System for J2SE and J2EE
- Small memory footprint (approximately 1MB)
- Multiple connections, multi-threaded architecture
- Ideal for traditional client – server and multi-tiered applications

Features and Benefits of PointBase Products
- 100% Pure Java™
- Near-zero administration
- SQL security and encryption
- Transaction management
- Fast time-to-market
- Low cost of development and deployment

For more information:
1-877-238-8798
(toll free US and Canada)
www.jp3.pointbase.com

PointBase®

(serial port Socket), which mimics the behavior of a Socket object to the application. The spSocket object talks through the serial port to an application running on the desktop that implements the pspSocket (proxy for serial port Socket) class. This target application is also written using the Waba classes. (The pspSocket class is accompanied by another class that, as you'll see, forms the outer loop for handling incoming requests. But because most of the work is done by pspSocket, I'll say that the desktop proxy implements that class.)

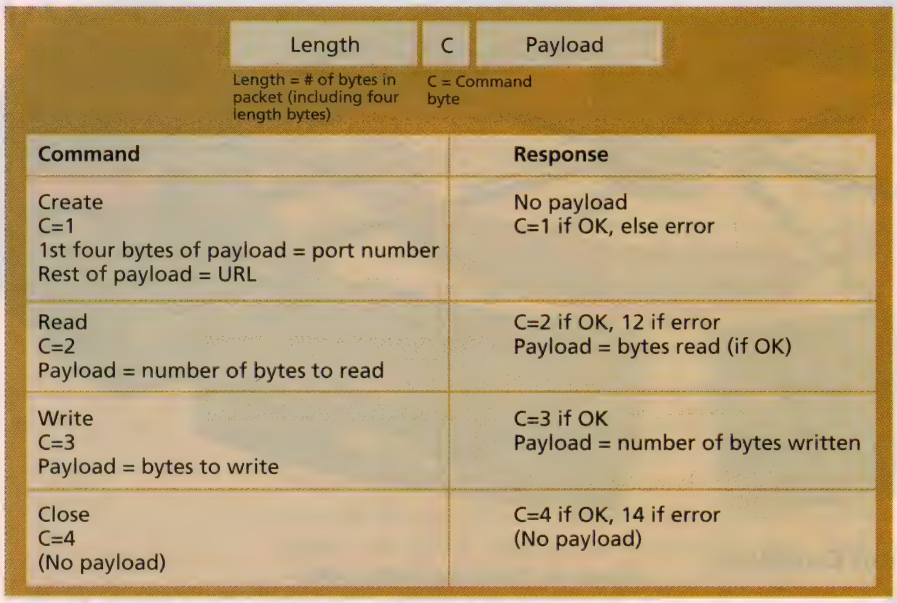
Simply put, requests that would ordinarily go to the Socket object go to the spSocket object, which stuffs them into a packet and sends them by serial port to the pspSocket application. The pspSocket application runs on a host (that does have Internet access), grabs the requests coming in from the client, passes them to a real Socket object, and sends the responses back.

The spSocket class provides all the important methods of Waba's Socket class:

- Constructor: The constructor not only instantiates a spSocket object, but also establishes a connection to the target URL and port.
- isOpen(): This method returns a boolean indicating whether the port was opened successfully.
- readBytes(): This method reads a byte array from the socket.
- writeBytes(): This method writes a byte array to the socket.
- close(): This method closes the spSocket. It closes not only the socket on the proxy side, but the serial port connection as well.

The only Socket method I haven't implemented is setReadTimeout(). I considered it unnecessary for the application I was

Continued on page 83.



Command	Response
Create C=1 1st four bytes of payload = port number Rest of payload = URL	No payload C=1 if OK, else error
Read C=2 Payload = number of bytes to read	C=2 if OK, 12 if error Payload = bytes read (if OK)
Write C=3 Payload = bytes to write	C=3 if OK Payload = number of bytes written
Close C=4 (No payload)	C=4 if OK, 14 if error (No payload)

Figure 2 | Packet Structure The protocol uses a simple variable-length packet scheme. A minimal packet includes a four-byte length field and a one-byte command field. Depending on the packet type, a payload of arbitrary length is appended.

Listing 1 Serial Port Socket

```
// class spSocket
// This is the client-side class.
// It implements a socket-through-
// a-serial-port
import waba.io.*;

public class spSocket
{
  // Commands
  static final byte CREATE = 1;
  // static final byte
  // CREATE_ERR = 11;
  static final byte READ = 2;
  // static final byte
  // READ_ERR = 12;
  static final byte WRITE = 3;
  // static final byte
  // WRITE_ERR = 13;
  static final byte CLOSE = 4;
  // static final byte
  // CLOSE_ERR = 14;

  // 10 sec timeout in millisecs
  static final int
  timeout = 10000;

  // Serial port object
  SerialPort sport;

  // Buffer to hold packet size and command
  byte[] phead;

  // Indicates open successful
  boolean open;

  // Create and open an spSocket
  public spSocket(String host,
    int port, int baud)
  {
    int i=0;
    open=false;

    // Open serial port 0
    sport = new SerialPort(
      0, baud);
    if(!sport.isOpen()) return;

    // Allocate packet header & fill it
    phead = new byte[5];
    intToByteArray(9+ host.length(), phead);
    phead[4]=CREATE;

    // Build payload buffer
    byte[] buf = new byte[4+host.length()];

    // Put port number in buffer
    intToByteArray(port,buf);

    // Copy host url to buffer
    for(;i<host.length();i++)
      buf[i+4]=(byte)host.charAt(i);
```

Continued on page 81.

The spSocket class represents the client side of the system. It implements a socket through a serial port. All the methods from the Socket class (with the exception of setReadTimeout()) are replicated.

Continued from page 80.

```
    // Flush the port
    flush();

    // Write the buffer
    sport.writeBytes(phead,0,5);
    sport.writeBytes(buf,0, buf.length);

    // Wait for reply.
    // It should be 5 bytes
    i=sport.readBytes(phead,0,5);
    if(i!=5) return;
    if(phead[4]!=CREATE) return;

    // All is OK
    open=true;
  }

  // Return true if socket is
  // open - false otherwise
  public boolean isOpen()
  {
    return(open);
  }

  // Read bytes from the spSocket
  // into a byte array
  public int readBytes(byte[] buf,
    int start, int count)
  {
    int i;

    // Flush the port
    flush();

    // Build and transmit request
    byte[] tbuf = new byte[4];
    intToByteArray(9,phead);
    phead[4]=READ;
    intToByteArray(count,tbuf);
    sport.writeBytes(phead,0,5);
    sport.writeBytes(tbuf,0,4);

    // Read response. First 5 bytes should be
    // count and command
    i=sport.readBytes(phead,0,5);
    if(i!=5) return(0);
    // Get the count and read the rest
    i=byteArrayToInt(phead,0);
    if(i==5) return(0);
    if(phead[4]!=READ) return(0);
    i=sport.readBytes(buf,start,
      i-5);
    return(i);
  }

  // Write bytes from the spSocket
  // into a byte array
  public int writeBytes(
    byte[] buf, int start, int count)
  {
    int i;

    // Flush the port
    flush();

    // Build header
    intToByteArray(count+5,phead);
    phead[4]=WRITE;

    // Send packet
    sport.writeBytes(phead,0,5);
    sport.writeBytes(buf,0,count);

    // Get response - should be 9-byte packet
    byte[] tbuf = new byte[9];
    i=sport.readBytes(tbuf,0,9);
    if(i!=9) return(0);
    if(tbuf[4]!=WRITE) return(0);
    return(byteArrayToInt(tbuf, 5));
    // All ok
  }

  // Close the spSocket
  public boolean close()
  {
    int i;

    // Flush the port
    flush();

    // Build packet
    intToByteArray(5,phead);
    phead[4]=CLOSE;

    // Send packet
    sport.writeBytes(phead,0,5);

    // Get response - should be 5 bytes
    i=sport.readBytes(phead,0,5);
    if(i!=5) return(false);
    if(phead[4]!=CLOSE)
      return(false);

    // All OK - close port
    sport.close();
    phead=null;
    open=false;
    return(true);
  }

  // Flush any input
  private void flush()
  {
    byte[] tbuf = new byte[2];

    // Set timeout for instant return
    sport.setReadTimeout(0);

    while(sport.readCheck()!=0)
      sport.readBytes(tbuf,0,1);

    // Return timeout to normal
    sport.setReadTimeout(timeout);
  }

  // Conversion routines
  // Convert int to 4-byte array
  private void intToByteArray(
    int val, byte[] buf)
  {
    int i, j;
    for(i=0;i<4;i++)
    {j = val & 255;
      buf[i] = (byte)j;
      val >>= 8;
    }
  }

  // Convert 4-byte array to int
  private int byteArrayToInt(
    byte buf[],int start)
  {
    int i=3;
    int j=0;
    for(;i>=0;i--)
    {j<<=8;
      j+=((int)buf[i+start])&255;
    }
    return(j);
  }
}
```

Listing 2 Serial Port Socket Proxy

```
// pspSocket.java
// Proxy for spSocket
import waba.io.*;

public class pspSocket {

  // Commands
  static final byte CREATE = 1;
  static final byte CREATE_ERR = 11;
  static final byte READ = 2;
  // static final byte
  // READ_ERR = 12;
  static final byte WRITE = 3;
  // static final byte
  // WRITE_ERR = 13;
  static final byte CLOSE = 4;
  static final byte CLOSE_ERR = 14;

  Socket socket;
  public byte[] pheader;
  public byte[] ipacket;
  SerialPort sport;

  // Constructor
  public pspSocket(int baud)
  {
    // Allocate header packet
    pheader = new byte[5];

    // Open the serial port
    sport = new SerialPort(1,baud);
  }

  // Verify ser. port is open
  public boolean isOpen()
  {
    return(sport.isOpen());
  }

  // Read header
  public int readpHeader()
  {
    int j;
    if(sport==null) return(0);
    j=sport.readBytes(pheader,0,5);
    return(j);
  }

  // Read payload
  public int readPayload()
  {
    // Get the payload length
    int len=byteArrayToInt(pheader,0)-5;
    if(len<=0) return(len);

    // Allocate for the payload
    ipacket = new byte[len];

    // Read the packet
    sport.readBytes(ipacket,0,len);
    return(len);
  }

  // Process create command
  public void pCreate()
  {
    // Get packet size
    int len = byteArrayToInt(pheader,0);

    // Get the port #
    int port = byteArrayToInt(ipacket,0);

    // Get the url
    String url = byteArrayToString(
      ipacket,4,len-9);

    // Create a socket
    socket = new Socket(url, port);

    // Return results
    // We'll use the pheader to return results
    intToByteArray(5,pheader,0);
    if(socket.isOpen())
      pheader[4]=CREATE;
    else
      pheader[4]=CREATE_ERR;
    sport.writeBytes(pheader,0,5);
  }

  // Process readBytes command
  public void preadBytes()
  {
    // Get # of bytes to read and
    // allocate a buffer
    int len = byteArrayToInt(pheader,0);
    byte[] rbuf = new byte[len];

    // Read the bytes
    len=socket.readBytes(rbuf,0,len);

    // Build a response packet
    // header in pheader
    intToByteArray(len+5,pheader,0);
    pheader[4]=READ;

    // Send header and payload
    sport.writeBytes(pheader,0,5);
    if(len>0)
      sport.writeBytes(rbuf,0,len);
  }

  // Process writeBytes command
  public void pwriteBytes()
  {
    // Get the # of bytes to write
    int len = byteArrayToInt(pheader,0)-5;

    // Write the packet
    len=socket.writeBytes(ipacket,0,len);

    // Return the response
    intToByteArray(9,pheader,0);
    pheader[4]=WRITE;
    sport.writeBytes(pheader,0,5);
    intToByteArray(len,pheader,0);
    sport.writeBytes(pheader,0,4);
  }

  // Process close command
  public void pclose()
  {
    boolean val;

    // Close the socket
    val=socket.close();
    socket=null;

    if(val)
      pheader[4]=CLOSE;
    else
```

Continued on page 83.

These two classes—pspSocket and pspSocketMain—implement the server side. The pspSocket class is the proxy for the spSocket; the former carries out the requests of the latter. The pspSocketMain class is not much more than an infinite loop that watches the serial port for incoming packets. When a packet arrives, pspSocketMain decodes the header and transfers control to the proper method to handle the request.

Continued from page 80.

crafting. For the sake of completeness, it could be implemented as a do-nothing method on the client side. (Or it could be implemented on the host side with minimal work. It would amount to adding a new command packet—I'll describe those in a moment—and adding code on the host to pass the request to the Socket object.)

Client and server communicate using the simple, variable-length packet scheme shown in Figure 2. Each packet begins with a five-byte header. The first four bytes of the header encode the packet length. (The packet length counts itself. So, the smallest possible packet would have a length count of five, because a header is the smallest possible packet.) Following the length is a single command byte. The protocol recognizes four commands, each corresponding to one of the methods implemented by spSocket.

Continued from page 82.

```
      pheader[4]=CLOSE_ERR;

    sport.writeBytes(pheader,0,5);
  }

  // Conversion routines
  // Convert int to 4-byte array
  private void intToByteArray(
    int val, byte[] buf, int start)
  {
    int i, j;
    for(i=0;i<4;i++)
    {j = val & 255;
      buf[i+start] = (byte)j;
      val >>= 8;
    }
  }

  // Convert 4-byte array to int
  private int byteArrayToInt(byte buf[],int start)
  {
    int i=3;
    int j=0;
    for(;i>=0;i--)
    {j<<=8;
      j+=((int)buf[i+start])&255;
    }
    return(j);
  }

  // Convert byte array to String
  private String
    byteArrayToString(byte buf[],int start, int len)
  {
    char cbuf[] = new char[len];
    int i=0;
    for(;i<len; i++)
      cbuf[i]=(char)buf[i+start];
    String s = new String(cbuf);
    return(s);
  }
}

// pspSocketMain.java
// Main class for spSocket proxy
import waba.io.*;
import waba.ui.*;
import waba.fx.*;
import waba.sys.*;

public class pspSocketMain extends MainWindow
{

  pspSocket mypspSocket;
  Graphics g;
  boolean c;
  int j;
  String s;

  public pspSocketMain()
  {
  // Instantiate the pspSocket and open the
  // serial port at 19200 baud
  mypspSocket = new pspSocket(19200);
  if(!mypspSocket.isOpen())
      exit(0);
    g = new Graphics(this);
    c=true;
    s=new String("X");

    // See if there's anything to be done
    while(true)
    {
      j=mypspSocket.readpHeader();
      if(j!=5)
      {
        if(c)
        {doClear();
          // Display idle message
          g.drawText("Idle",0,1);
          c=false;
        }
        continue;
      }

      // Read the payload
      mypspSocket.readPayload();
      doClear();
      c=true;

      // Get the command and act accordingly
      switch(mypspSocket.pheader[4])
      {
        case pspSocket.CREATE:
          g.drawText("Create",0,1);
          mypspSocket.pCreate();
          break;

        case pspSocket.READ:
          g.drawText("Read  ",0,1);
          mypspSocket.preadBytes();
          break;

        case pspSocket.WRITE:
          g.drawText("Write ",0,1);
          mypspSocket.pwriteBytes();
          break;

        case pspSocket.CLOSE:
          g.drawText("Close ",0,1);
          mypspSocket.pclose();
          break;

        default:
          break;
      }
      doClear();
    }
  }

  public void doClear()
  {
    g.setColor(255,255,255);
    g.fillRect(0,0,100,30);
    g.setColor(0,0,0);
  }
}
```

After the packet header comes an optional payload, whose length and content varies depending on the command. For an open operation, the payload carries the target port number in the first four bytes, followed by the string of the URL to be opened (encoding into a byte array). A read bytes packet has only a four-byte payload: the byte-encoded integer value representing the number of bytes to be read from the Socket. A write bytes packet's payload consists of the stream of bytes to be written to the Socket. (The number of bytes to be written is, of course, encoded in the packet's header.) Finally, the close operation packet has no payload; it consists of only a four-byte count and a single command byte.

The host proxy responds to requests with a similar packet format. The response to an open packet is simply a five-byte packet header. The command field is used to indicate whether the open was successful. The response to a request to read bytes can vary, depending on whether the read was successful. If unsuccessful, the read operation will return only a five-byte response (only a byte count, an error value in the command field, and no payload). If successful, the read operation will return a packet with a payload containing the bytes read from the Socket. (The number of bytes read might be less than the number of bytes requested to be read. But, the client can recognize that simply by looking at the packet length.)

The packet structure of a response to a request to write bytes is simply a packet header: a byte count, followed by a code indicating success or failure. The same structure is used in the response packet to a close request.

The first thing to note as you look through the source code for spSocket is that I've commented out some of the response command finals (see Listing 1). The proxy indicates a successful execution of a command by placing the same command code in the response packet. So the spSocket class methods simply recognize an error by a response command code that does not match the one sent.

The methods are fairly straightforward; all method prototypes look like their Socket counterparts. I did have to modify the constructor and add a baudrate parameter. I hardcoded the timeout for the serial port to 10 seconds, which seemed more than enough time and worked fine in all my tests. (If you decide to create a setTimeOut() method, you'll have to figure out which timeout is being set—the socket's or the serial port's. I'll leave that for someone else to do.) Each method simply creates a request packet, transmits it to the proxy, gathers the response, and returns it to the application. As you'll see shortly, if your application already uses the Socket class, very little modification is needed for it to run the spSocket class.

The server-side code consists of two classes: pspSocket and pspSocketMain (see Listing 2). The pspSocket class is the server counterpart to spSocket. Methods in pspSocket have a near one-to-one correspondence to methods in spSocket. Each pspSocket unpacks a request packet, calls the proper Socket method, retrieves the result, packages the result into a response, and ships it back to the client.

The pspSocketMain class is the proxy's outer loop. It executes forever, looking for input from the serial port. When pspSocketMain receives a packet header, it reads and decodes the header, reads the payload, and dispatches control to the appropriate pspSocket method. The pspSocketMain class also provides "I'm alive" output by displaying a string that indicates which command is being processed.

Easy Modification

Modifying an application to use spSocket is reasonably simple. I'll use the Web application that is available with the Waba download as an example. This application consists of two classfiles, Web and WebPage. The only file that you have to change is WebPage.java.

First, toward the top of the application where class-wide variables and objects are defined, change the Socket object to an spSocket object. Next, in the load() method, find the location where the Socket object is instantiated. Change the constructor call to:

```
socket = new
  spSocket(host,80,19200);
```

and comment out the following line (which sets the read timeout). Finally, locate the prototype for the socketWriteln() method, and change the datatype of its first parameter from Socket to spSocket.

Place the spSocket.java file in the same subdirectory as the Web application and run the supplied make.bat file. The batch file is coded so that it bundles all the Java code into the final PDB and PRC files for downloading to the Palm. Download the result to the Palm (make sure you have the Waba VM installed). Compile the pspSocket.java and pspSocketMain.java files, and launch them under the appropriate desktop version of the Waba VM. (Because I run a Windows 2000 system, I used the SuperWaba Windows VM, available from www.superwaba.org.) Launch your newly created Web application, touch the Go button, and . . . well . . . go!

If You Want It Neat

I did discover another way to skin this particular cat. It's far neater and cleaner than the example I've just shown, and it doesn't require you to recode any applications on handhelds that use Waba's Socket class. This neater solution is called Mocha W32 PPP. It allows a Palm or Windows CE device to connect to a Windows 9x system (I also tested it on Windows 2000) through the serial port. It provides a PPP connection through the serial port.

Mocha W32 PPP is neater because when it's running, it puts up a much nicer window than does my solution. Best of all, with Mocha W32 PPP, you don't need to do any special coding. Also, it's configurable. You can, for example, set a variety of baudrates and choose a different serial port (pspSocketMain is hardwired to COM1), and it even cooperates with Hotsync (you have to turn off Hotsync to use pspSocketMain).

Mocha W32 PPP is available at www.mochasoft.dk, where you can download a trial version. A single-user license is only $9.95. Check it out. JP

GO ONLINE
www.javapro.com

Use these Locator+ codes at www.javapro.com to go directly to these related resources.

JP0201 Download all of the code for this issue of *Java Pro*.

JP0201JG Download the code for this article separately.

JP0201JG_T Read this article online.

Unique code, enhanced content, interviews and archives

Get an Extra Shot of

Java Pro

New Web Site!

Java Pro is now online at **www.javapro.com!** Check it out for enhanced content from the magazine, code, original articles, and interviews with the pioneering minds shaping the Java development community.

Point your browser to www.javapro.com and find:

- Articles from the current issue of *Java Pro* —code, extra resources and content!
- Comprehensive archives going back to 1997!
- Interviews with leading Java authorities!
- And much more!

While you're there, be sure to sign up for a FREE customized subscription to *Java Insight*, the weekly e-mail newsletter that brings you the best Java resources, news, and information from *Java Pro* and the Web at large.

Check it all out at:

www.javapro.com

FTP FAWCETTE TECHNICAL PUBLICATIONS

Java is a trademark of Sun Microsystems.
Java Pro is published by Fawcette Technical Publications, Inc.

Continued from page 23.

to the hidden object (in this case, a remote OrderService object). The invoke() method has three arguments: the proxy object, the method being called, and its arguments. The proxy object is of no use in this context. I use the method being called and its arguments to generate the SOAP request. Ignore the encoding style and caching for now (I'll elaborate on them later). The serialization of the arguments and of the return type needs some explanation. Serialization is handled by the Apache SOAP BeanSerializer class, which takes care of the serialization of an arbitrary class, provided it behaves like a JavaBean (in other words, the class must define a public default [empty] constructor and set/get methods that can be used to serialize the object). The Apache SOAP implementation requires the registration of mapping entries that essentially define the name of the class that serializes your object. A more flexible implementation would extract the mappings from a deployment file to support arbitrary classes without modifying the SavonProxy implementation. Note that BeanSerializer automatically handles Java data types.

Now that you've set up the call, you can invoke the SOAP service via the Call.invoke() method (not to be confused with the SavonProxy.invoke() method). The Apache SOAP implementation generates the SOAP request, posts it to the SOAP service, deserializes the response, and creates the objects for the response. The invoke() method throws a SavonException object when something goes wrong; for instance, the server could return an HTTP error or a SOAP:FAULT section. An example of a SOAP request and a SOAP response are available online at www.javapro.com

From the point of view of the calling application, only one method call has been coded (see Listing 4). This level of simplicity is a significant improvement over "hand-coded" creation and parsing of XML documents. The cost of this simplicity is in the dynamic proxy and the SavonProxy.invoke() method. Arguably, the few instructions contained in that method constitute a negligible cost compared to the latency associated with a network, even in the case of an intranet. Local implementation requires no proxy.

The methods defined in the Orders interface take a user name and password that could be used to authorize and authenticate the user. The implementation I've provided ignores the username and password.

The Orders.getOrderList() method differs from the Orders.runOrder() method in that the latter returns an XML document. You can return an XML document in one of two ways: as a String or as an org.w3c.dom.Element. When you return an XML document as a String, special characters such as < and > must be escaped (for instance, < is escaped into <). The major drawbacks of this approach are the increased size and poor readability of the document. For those reasons, I prefer to use the org.w3c.dom.Element methodology whenever possible.

> SOAP can ride with other HTTP traffic, such as Web pages, and SOAP doesn't require the network administrator to perform any additional setup or maintenance

I've packaged the Le Savon application as a Web project that generates a class tree for local debugging and a WAR file to deploy with a J2EE server (at www.javapro.com). I've included the project file for JBuilder 4 and Apache Ant. I've built everything on a z:\ drive, so you'll have to substitute a drive or edit the Ant files to build on your machine. You'll need to register the service (either manually or using register.bat) before running. You'll also need to make sure that Tomcat can see your classes; for instance, you could include the path to lesavon\WEB-INF\classes in CLASSPATH in tomcat.bat around line 114.

Special Considerations

Debugging Debugging remote services like JSP and servlets tends to be a tedious task because of the lack of support by development tools and the latency involved in remote debugging. The SavonProxy implementation offers an alternative, because you can debug the service as a local object simply by changing the URL. The remainder of the code is identical for local and remote invocations (see Listing 4).

When your service works in the local environment, you'll want to debug the SOAP messages that go across the wire. A practical approach is to use the tcpTrace utility that comes with Apache SOAP. The tcpTrace utility sets up a tunnel between two ports. For instance, if you install Tomcat on port 8080, you can set up tcpTrace to listen to port 8081 and echo to port 8080 in order to view all HTTP requests as they come along.

Flexibility The actual load of a Web application tends to be hard to predict. The problem becomes more complicated when the Web application is sold to many customers as opposed to being deployed on a specific Web site. By switching easily from a remote to a local deployment, the SavonProxy implementation offers flexibility in that regard.

An application using this technology could read the URLs of its services from a deployment file that a site administrator could change easily based on the number of servers available and the load of the site. Because there's no penalty for local calls, this flexibility doesn't compromise the performance of the application.

If high availability of the service is a requirement, a background thread could test the remote service and switch to local when things go wrong with the remote service. You could use the same methodology to maximize the use of hardware: Work with local calls when the load is low and start distributing the requests when the load increases.

Caching Web servers often serve a large community of users with similar needs. For instance, a business-intelligence application can provide identical key performance indicators to many users. Similarly, a traffic-report system can return identical data for the same city. In the case of Le Savon, several employees from client companies might request the same information. Typically, Web servers offer improved performance when they can cache the data they return to users. Caching is straightforward in the case of static data such as Web pages but can be more challenging with dynamic data. SOAP

Listing 2 Return Orders

```
/*
 * OrderService.java    1.0 02/06/2001
 *
 */
package com.lesavon.service;

import java.io.*;
import java.net.*;
import java.util.*;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.
  ParserConfigurationException;

import org.w3c.dom.*;
import org.xml.sax.*;
import org.apache.soap.util.xml.*;

import com.lesavon.*;

/**
 * Orders interface implementation
 * <p>
 * This class is the actual implementation of
 * the Orders service.
 * It gets the real data from the platform
 * server. The data might be cached.
 * </p>
 * @author Henry Bequet
 * @version 1.0 02/06/2001
 * @since 1.0
 * @see Orders
 */
public class OrderService implements
  com.lesavon.Orders {
  // All users have the same to orders
  Order orders[] = new Order[]
  { new Order("First Order", "Savon de lilas"),
      new Order("Second Order",
      "Savon de lavande")
  };

  /**
  * Get a list of Orders for a user.
  *
  * @param username the name of the user for whom
  * we are getting the list of Orders
  * @param password her password
  * @return the list of Orders
  */
  public Order[] getOrderList(String userName,
    String password)
  throws SavonException {
    return orders;
  }

  /**
  * Get the specified order.
  *
  * @param username the name of the user for whom
  * we are returning the orders
  * @param password her password
  * @param orderId the id of the order to return
  * @return the XML of the rrder
  */
  public Element getOrder(String userName,
    String password, String orderId) throws
    SavonException {
    for(int index = 0; index < orders.length;
      index++) {
      if(orders[index].getId().equals(
        orderId)) {
        try {
          XMLParserLiaison xpl =
            new XercesParserLiaison();
          Document document = xpl.read (
            "Le Savon Order",
            new StringReader(
            orders[index].toXML()));
          return document.getDocumentElement();
        } catch (Exception exception) {
          System.out.println(
            "Exception: " + exception);
          throw new SavonException(
            "Cannot create XML response",
            exception);
        }
      }
    }

    throw new SavonException(
      "Order " + orderId + " not found!");
  }
}
```

The Orders.getOrderList() method in the OrderService class returns an array of orders.

Listing 3 Savon Proxy

```
/*
 * SavonProxy.java 1.0 02/06/2001
 *
 */
package com.lesavon.proxy;

import java.lang.reflect.*;
import java.io.*;
import java.net.*;
import java.util.*;
import org.apache.soap.util.xml.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;
import org.apache.soap.util.net.TcpTunnelGui;
import org.apache.soap.encoding.*;
import org.apache.soap.encoding.soapenc.*;
import com.lesavon.*;
import com.lesavon.cache.SOAPCache;

/**
 * A dynamic proxy to be used as the SOAP proxy
 * for a SOAP provider. There is actually no
 * target in this proxy since the
 * call is meant to be serialized to a SOAP
 * server.
 * @author Henry Bequet
 * @version 1.0 02/06/2001
 * @since 1.0
 */
public class SavonProxy implements
  InvocationHandler {
  static XMLParserLiaison xpl=
    new XercesParserLiaison ();

  private URL savonURL;
  private Call call;
```

Continued on page 88.

The important code in the SavonProxy class can be found in the invoke() method, which traps all the calls going to the remote OrderService object.

Continued from page 87.

```
    private SavonTransport transport;
    private String targetClass;

    /**
    * Constructor.
    * @param targetClass the class (name) of the
    * target
    * @param url the URL of the service
    */
    public SavonProxy(
       String targetClass, URL url) {
       call = new Call();
       call.setTargetObjectURI("urn:orders");
       call.setSOAPTransport(transport =
          new SavonTransport());
       this.targetClass = targetClass;
       this.savonURL    = url;
    }

    /**
     * Constructs an instance of the target
     * interface for localhost references or of the
     * target interface proxy for remote
     * implementation.
     * @param url    the target host
     * @param targetInterface the interface pointer
     * to return
     * @param targetClass the implementation of the
     * targetInterface
     * @return an instance of the target interface
     * implementation or of the proxy for remote
     * urls
     * @throws ClassNotFoundException if the class
     * cannot be located
     * @throws IllegalAccessException if the caller
     * does not have the permission to load the
     * class
     * @throws InstantiationException if the class
     * cannot be instantiated (if it is an
     * interface or an abstract class)
     */
    public static Object newInstance(String url,
       String targetClass)
       throws ClassNotFoundException,
       IllegalAccessException,
       InstantiationException,
       MalformedURLException {
       Object result = null;

       if (url.startsWith("inproc:")) {
          result =
             Class.forName(targetClass).newInstance();
          } else {
             result = Proxy.newProxyInstance(
                Class.forName(
                targetClass).getClassLoader(),
                Class.forName(
                targetClass).getInterfaces(),
                new SavonProxy(
                targetClass, new URL(url)));
             }

          return result;
       }

       public Object invoke(
          Object proxy, Method method,
          Object[] args)
          throws Throwable {
          Object result = null;

          try {
             // Encoding is NS_URI_SOAP_ENC,
             // except if the method returns a DOM
             Class elementClass =
                Class.forName("org.w3c.dom.Element");
             String callEncoding =
                elementClass.equals(
                method.getReturnType())
                ? Constants.NS_URI_LITERAL_XML
                : Constants.NS_URI_SOAP_ENC;
             String paramEncoding =
                elementClass.equals(
                method.getReturnType())
                ? Constants.NS_URI_SOAP_ENC
                : null;

             // Build the call.
             call.setMethodName(method.getName());
             call.setEncodingStyleURI(
                callEncoding);
             Vector params = new Vector ();
             for (int index = 0; index <
                args.length;
             index++) {
                params.addElement (new Parameter(
                   "arg" + index,
                   method.getParameterTypes()[index],
                   args[index], paramEncoding));
                }
                call.setParams(params);

                SOAPMappingRegistry smr =
                   new SOAPMappingRegistry();
                BeanSerializer beanSer =
                   new BeanSerializer();

                smr.mapTypes(
                   Constants.NS_URI_SOAP_ENC,
                   new QName("urn:lesavon-order",
                   "order"), Order.class, beanSer,
                   beanSer);
                call.setSOAPMappingRegistry(smr);

                // We set the cache key (optional,
                // this frees the server from
                // computing it)
                transport.setCacheKey(
                   SOAPCache.computeCacheKey(
                   targetClass, method, args));
                Response resp = call.invoke(
                   savonURL, "");

                String responseString =
                   call.getEnvelopeString(transport);
                SOAPContext context =
                   resp.getSOAPContext();
                ByteArrayOutputStream stream =
                   new ByteArrayOutputStream(1024);
                context.writeTo(stream);
                responseString  = stream.toString();

                // Check the response.
                if (resp.generatedFault()) {
                   Fault fault = resp.getFault();
                   throw new SavonException(fault);
                } else {
                   result =
                      resp.getReturnValue().getValue();
                   }
                } catch (Exception exception) {
                   throw exception;
                }

                return result;
             }
          }
```

provides an easy way to define a key for a request—the serialization of the input arguments. That serialization is readily available because the Apache SOAP implementation uses it to generate the SOAP request.

The SavonProxy.invoke() method computes the cache key and adds it to the header of the HTTP request via the SavonTransport.setCacheKey() method. The advantage of computing the cache key on the client is that the server can simply look at the HTTP header and, if the cache key is present, improve performance by not parsing the request. If the cache key is not present, the server must parse the request to compute the cache key. The optional presence of the cache key in the HTTP header allows the server to support clients that provide the optimization and clients that don't. When the request isn't present in the cache, the server can simply forward the request to the SOAP service for processing. If thousands of users hit the site with similar requests, not parsing requests on the server saves many CPU cycles.

The SavonTransport class extends the SOAPHTTPConnection class to override the send() method and add the cache key to the HTTP header. You populate the cache by saving the response from the service to a file, and the key along with the filename in a map. A production cache would probably include features such as selective additions to the cache, and cache operations such as flushing.

One caveat: Because I use the RequestDispatcher to forward requests from the cache to the SOAP router servlet, I must include the Apache SOAP router as part of the application, not as its own servlet: It's not legal to forward or include the response of a servlet in another context. An alternative implementation would be to subclass the SOAP router servlet. This would perform better when the cache can't be used, but at the expense of flexibility—the SOAP router servlet must run on the same machine as the cache.

Scalability The issues of performance and scalability always come up when you're developing a Web server. Most of the discussion so far has focused on the client, where the issue of scalability is minimal. The problem is how well a SOAP implementation will scale. By definition a SOAP system is stateless (no client state is maintained from call to call), so a well-crafted implementation will scale, especially if it can use a cache similar to what I described earlier. The lack of state is usually a good recipe for scalability because objects created for one request can be reused for another request without paying the penalty of saving and restoring state. Finally, with the lack of state, throughput should be linear with the addition of CPUs and/or servers.

When you're using a SOAP implementation based on servlets (like the Apache SOAP implementation), a great deal of the scalability comes from your servlet engine. Therefore, carefully choosing and tuning your application server is critical.

Limitations

An application based on SOAP has some limitations, especially compared to full-fledged distributed-object systems like RMI or DCOM. For instance, all method calls are stateless in the sense that the remote object is available only for the time of the call. In other words, there is no "live" connection to a remote object. This limits somewhat the flexibility of the remote interface: Methods must be thought of as "static" because one function call can't set a state in the remote object and then have that state retrieved by other calls. This limitation might be annoying at times because it forces the interface designer to decide between a shallow and a deep implementation when returning object hierarchies. When a choice can't be made, you might have to define two methods. This brings up another limitation: Because the interface is defined in terms of Java methods, overloading based on the return type is forbidden. Finally, all objects are passed by value and only one object can be returned by a method call. However, the returned object can be an array or a complex object.

> If you need a royalty-free solution you can deploy relatively easily, you might consider a SOAP-based solution

The implementation I've presented is far from complete, but you can use the ideas in your application to implement a distributed architecture based on SOAP. Some of the program could use a little refinement. For example, you could develop a more sophisticated caching system that allows specific arguments to be used in the computation of the cache key, and that supports a time to live based on the method called and perhaps the argument values. Also, a more flexible registration of the mapping entries for serialization would improve the extensibility of the system.

You might be wondering by now why I don't recommend simply working with a J2EE server and not worrying about these problems. The solution I've presented here is simple, low-cost, and open. Application servers are expensive to deploy and require significant maintenance. If you need a royalty-free solution you can deploy relatively easily, you might consider a SOAP-based solution. In addition, if you need a solution that can work in a distributed environment with non-Java components, SOAP might be just what you need. JP

Listing 4 Get a List

```
Orders ordersRemote =
  (Orders)SavonProxy.newInstance(
  "http://localhost:8081/LeSavon/servlet/LeSavon",
  "Orders", "OrderService");
Orders ordersLocal =
  (Orders)SavonProxy.newInstance(
  "inproc://", "Orders", "OrderService");

Order localOrders[] =
  ordersLocal.getOrderList(user, password);
Order remoteOrders[] =
  ordersRemote.getOrderList(user, password);
```

Notice that, in the code for getting a list of orders, the local and remote calls are identical.

GO ONLINE
www.javapro.com

Use these Locator+ codes at www.javapro.com to go directly to these related resources.

JP0201 Download all of the code for this issue of *Java Pro*.

JP0201HB Download the code for this article separately.

JP0201HB_T Read this article online.

XML Messaging with JAXM

A crucial Java piece in the Web services scheme, JAXM brings some structure to XML messaging

by Daniel F. **SAVARESE**

The components for building Web services in Java are coming together slowly, but they've still got a long way to go. The Java API for XML Processing (JAXP) was released first, then the Java Architecture for XML Binding (JAXB), and now the Java API for XML Messaging (JAXM), in some ways the most crucial piece, is ready. The haste with which some of these APIs were put together is evident when you start using them, but that's what 2.0 revisions are supposed to fix. In the meantime, you can build Web services with them more easily than without, but not as easily as you might like.

As implied by its name, JAXM facilitates the sending and receiving of XML messages to and from Web services. Although it has the potential to support multiple messaging standards, JAXM requires only that the Simple Object Access Protocol (SOAP) and SOAP with Attachments be supported by conformant implementations. However, SOAP is a very general messaging framework and does not dictate the content or meaning of message headers and bodies. Therefore JAXM also supports the notion of SOAP profiles, which are industry-defined standards that specify the required contents and meaning of message headers and bodies. For example, ebXML defines how addressing and message identification information should be specified in a SOAP message.

JAXM can be divided into two categories: message assembly and communication. Message assembly is handled by the classes in the javax.xml.soap package. Communication is handled by the javax.xml.messaging package. Both packages contain so few classes that you should be able to grasp most of JAXM just by reading the javadocs. This may be why JAXM is billed as a "lightweight messaging API" in the specification. Don't be fooled. The JAXM reference implementation requires DOM4J, Log4J, JAXP, JavaMail, and the JavaBeans Activation Framework to work. That is a far cry from being "lightweight" and just goes to show that the term is overused.

> JAXM is billed as a "lightweight messaging API" in the specification. Don't be fooled

The javax.xml.soap package takes some of the pain out of creating SOAP messages. Even though SOAP messages are XML documents and can be created using DOM, the JAXM SOAP classes recognize the structure of SOAP messages and present a simpler means of working with SOAP. SOAP messages are represented by the SOAPMessage class. Every SOAP message has a SOAP part, corresponding to the original SOAP specification, and an attachment, corresponding to the attachments extension. The SOAP part contains an envelope that encloses a message header and body. Each of these elements is modeled by a separate class or interface, including SOAPPart, AttachmentPart, SOAPEnvelope, SOAPHeader, and SOAPBody. The elements of a SOAP message form a tree, with each node implementing the Node interface and, in

Listing 1 Web Service Client

```
package example;

import java.io.*;
import java.util.*;

import javax.xml.messaging.*;
import javax.xml.soap.*;

public class QuoteClient {
  URLEndpoint service;
  static SOAPMessage cannedQuery;

  static {
    try {
      MessageFactory factory =
        MessageFactory.newInstance();
      SOAPBody body;
      SOAPEnvelope envelope;
      Name name;
      SOAPElement child, stock;

      cannedQuery = factory.createMessage();
      envelope =
        cannedQuery.getSOAPPart().
        getEnvelope(true);
      envelope.addNamespaceDeclaration(
        "stock", "http://www.stocksrus.com/stock");
      body = envelope.getBody();

      name = envelope.createName(
        "GetStockPrices", "stock");
      child = body.addChildElement(name);
      name = envelope.createName(
        "stocks", "stock");
      child = child.addChildElement(name);
      name = envelope.createName(
        "stock", "stock");
      stock = child.addChildElement(name);
```

Continued on page 91.

JAXM allows you to write standalone Web service clients, like the one shown here, or clients that run as servlets.

Continued from page 90.

```
        name = envelope.createName(
          "symbol", "stock");
        stock.addChildElement(name).
          addTextNode("TYR");
        name = envelope.createName(
          "stock", "stock");
        stock = child.addChildElement(name);
        name = envelope.createName(
          "symbol", "stock");
        stock.addChildElement(
          name).addTextNode("ELL");
        stock = child.addChildElement(name);
        name = envelope.createName(
          "stock", "stock");
        stock = child.addChildElement(name);
        name = envelope.createName(
          "symbol", "stock");
        stock.addChildElement(
          name).addTextNode("HAL");
        cannedQuery.saveChanges();
      } catch(SOAPException se) {
        se.printStackTrace();
        System.exit(1);
      }
    }
```

```
    public QuoteClient(String url) {
      service = new URLEndpoint(url);
    }

    public static void printCannedQuery(
      PrintStream out)
      throws SOAPException, IOException
    {
      out.println("\nSOAP Query:\n");
      cannedQuery.writeTo(out);
      out.println();
    }

    public void getQuote(PrintStream out)
      throws SOAPException, IOException
    {
      SOAPConnection connection;
      SOAPMessage reply;

      connection = SOAPConnection.newInstance();
      reply = connection.call(cannedQuery, service);

      out.println("\nReply:\n");
      reply.writeTo(out);
```

Continued on page 92.

most cases, also the SOAPElement interface. A SOAPMessage can be built up by adding child elements to each node, much like DOM.

A SOAPMessage is created with a MessageFactory. In turn, a MessageFactory can be created by calling the static MessageFactory.newInstance() method. Alternatively, you can create a ProviderConnection from the javax.xml.messaging package and call its createMessageFactory(String) method. When you create a MessageFactory from a ProviderConnection, you have to specify a profile name. The resulting MessageFactory will be able to create SOAP messages that conform to that profile. To complicate matters, a ProviderConnection can be created only from a ProviderConnectionFactory. A ProviderConnection bridges a JAXM application to the container in which it is running. JAXM applications may run as either standalone programs or inside containers. Applications that run inside containers (for example, as a servlet) can make use of run-time support in the form of a JAXM provider. A provider can regulate access to message factories, but can also provide services such as asynchronous message delivery. For example, a standalone application that wished to receive an asynchronous reply to a message would have to implement its own asynchronous message-handling system. An application running inside of a container can delegate that responsibility to the provider.

Listing 1 contains the code for a simple standalone JAXM client. It generates the SOAP message in Listing 2 (borrowed from "ONE Web to Bind Them" in the *Java Pro* special issue "Guide to Application Servers 2001"), which queries a Web service for the prices of a list of stocks. In this example, we opt to create a default MessageFactory using newInstance(). Then an empty message is instantiated with createMessage(). The SOAP part of the message is just a shell for the envelope, so

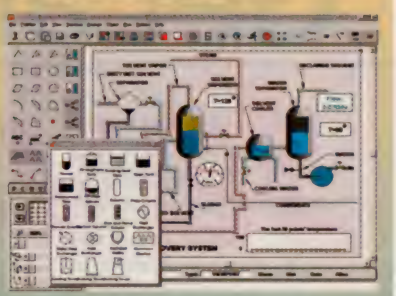
GLG Toolkit

Award-Winning Graphics System

Dynamic Visualization Framework for Java and C/C++

GLG is the most complete dynamic graphics framework that handles both ***Real-Time C/C++*** and ***100% Pure Java,*** offering breakthrough performance and productivity gains.

It's ideal for a range of applications; from building custom graphs and gauges to designing dynamic graphical displays for networking, telecommunications, process control, simulation and others.



Point-and-Click Graphics Builder for creating dynamic HMIs and custom components

Dynamic Graphics Framework

- Interactive Graphics Builder
- Dynamic animation and constraints
- Library of pre-built components
- A choice of ***100% Pure Java*** or ***Cross-Platform C/C++*** versions
- Java 2, ***Swing*** and ***Java2D*** support
- Winner of IBM 's ***Cool Tool*** and the ***JDJ Editor's Choice*** awards for the best Java bean
- Free Evaluation Copy

Dynamic GIS and Diagramming Capabilities

New Avionics Gauges

Pre-built and custom components, Virtual Instrumentation

Generic Logic, Inc.
Web: www.genlogic.com
Email: glg@genlogic.com
Call: 413/253-7491

© 2001 Generic Logic, Inc.

On-Line Demos: www.genlogic.com

Continued from page 91.

```
    out.println();
    connection.close();
  }

  public static class Main {
    public static final String DEFAULT_URL =
      "http://localhost:8080/quotes/getquote";

    public void run(String args[])
      throws SOAPException, IOException
    {
      QuoteClient client;
      String url;

      if(args.length > 0)
        url = args[0];
      else
        url = DEFAULT_URL;

      client = new QuoteClient(args[0]);

      QuoteClient.printCannedQuery(System.out);
      client.getQuote(System.out);
    }
  }

  public static void main(String args[]) {
    try {
      new Main().run(args);
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

Listing 2 SOAP Query

```
<Envelope xmlns="http://schemas.xmlsoap.org/
  soap/envelope/"xmlns:stock=
  "http://www.stocksrus.com/stock">
  <Header/>
  <Body>
    <stock:GetStockPrices>
      <stock:stocks>
        <stock:stock>
          <stock:symbol>TYR</stock:symbol>
        </stock:stock>
        <stock:stock>
          <stock:symbol>ELL</stock:symbol>
        </stock:stock>
        <stock:stock>
          <stock:symbol>HAL</stock:symbol>
        </stock:stock>
      </stock:stocks>
    </stock:GetStockPrices>
  </Body>
</Envelope>
```

This is a formatted version of the SOAP query generated by QuoteClient (in Listing 1).

Get an Extra Shot of

Java Pro

New Web Site!

- Articles from the current issue of *Java Pro* —code, extra resources and content!
- Comprehensive archives going back to 1997!
- Interviews with leading Java authorities!
- And much more!

Check it all out at:

www.javapro.com

FTP FAWCETTE TECHNICAL PUBLICATIONS

Java is a trademark of Sun Microsystems. *Java Pro* is published by Fawcette Technical Publications, Inc.

we create the envelope next. SOAPPart.getEnvelope(boolean) will return the existing envelope of a message if it is given an argument of *false* and will create a new envelope if it is given an argument of *true*. To disambiguate the meaning of the message, we create a namespace declaration in the envelope. Using namespaces allows multiple elements with the same name but different meanings to coexist in the same XML document. After creating the message body, the elements of the message can be added on a per-node basis with addChildElement(Name). Each element has a tag name that must be created from the envelope with createName(). If the tag name belongs to a namespace, its namespace can be specified as in the example, where all the tags belong to the stock namespace. Once the message has been assembled, you finalize the changes with saveChanges().

As you can see, assembling SOAP messages piecemeal is rather verbose. As a shortcut, it is also possible to take XML from another source, such as a file, and import it wholesale as a JAXP javax.xml.transform.Source instance using SOAP-

Listing 3 Web Service

```
package example;

import java.util.*;

import javax.xml.messaging.*;
import javax.xml.soap.*;

public class QuoteServlet extends JAXMServlet
  implements SyncListener
{
  MessageFactory factory;
  static HashMap quotes;

  static {
    quotes = new HashMap();
    quotes.put("TYR", "6");
    quotes.put("ELL", "4");
  }

  public QuoteServlet() throws SOAPException {
    factory = MessageFactory.newInstance();
  }

  public SOAPMessage onMessage(
    SOAPMessage message) {
    SOAPMessage reply = null;
    SOAPEnvelope envelope;
    SOAPBody body;
    SOAPElement getStockPrice, stocks, stock;
    Name name;
    Iterator child;

    try {
      envelope =
        message.getSOAPPart().getEnvelope();
      body = envelope.getBody();

      reply = factory.createMessage();
      envelope =
```

Continued on page 94.

The JAXMServlet class makes it easier to write Web services that process SOAP messages.

Advertiser Index

BEA Systems 800-817-4BEA www.bea.com	2	**IBM Corporation** www.ibm.com	30, 31	**Pramati Technologies** www.pramati.com	52	**VMGear** 888-655-0055 www.vmgear.com	62
BEA e-World 2002 www.bea.com	4	**ILOG** 800-FOR-ILOG www.ILOG.com	29	**Rational** www.rational.com/offer/scmguide14	17	**WebGain** 877-WebGain www.webgain.com/goto/jp.html	61
Borland www.borland.com/new/jb6/5002.html	15	**Infragistics, Inc.** 800-231-8588 www.infragistics.com	C4	**SilverStream** www.silverstream.com	6	**ZeroG** www.ZeroG.com/goto/ia45jp	25
Compuware Corporation www.compuware.com	68	**InterSystems Corporation** www.intersystems.com	27	**Sitraka Software** 888-361-3264 www.sitraka.com/serverchart1/jp	1		
Corda Technologies 888-763-0517 www.corda.com	13	**IONA Technologies** www.iona.com	8, 9	**Sitraka Software** 888-361-3264 www.sitraka.com/jclass/jp	57		
CSIRO www.cmis.csiro.au/sact	55	**Java Pro Insight Newsletter** www.javapro.com	67	**Sitraka Software** 888-361-3264 www.sitraka.com/jprobe/jp	C3		
Embarcadero Technologies www.embarcadero.com	21	**Java Pro Magazine Subscriptions** 800-848-5523 www.javapro.com	41	**Sonic Software** 866-GET-SONIC www.sonicsoftware.com/jp	48, 49		
Fiorano www.fiorano.com	39	**Java Pro Online** www.javapro.com	85, 92	**SpiritSoft** www.spiritsoft.net/downloads	11		
Fresher Information Corp. 415-536-5394 www.fresher.com	32	**Parasoft Corporation** 888-305-0041 www.parasoft.com/jpro1	36	**Togethersoft Corporation** 919-833-5550 www.togethersoft.com/1/jp.jsp	C2		
Generic Logic 413-253-7491 www.genlogic.com	91	**PointBase** 877-238-8798 www.jp3.pointbase.com	79				

Product Information

- To get immediate details on products advertised in this issue, you can simply log onto our online services at http://infoasap.devx.com.
- This provides a list of direct links to this month's advertiser's Web sites.

Advertising Sales Contacts

- Shana Deane, Advertising Director 413.665.6910, sdeane@fawcette.com
- Brian Levine, Eastern Regional Sales Manager 201.656.2688, blevine@fawcette.com
- Janice Mahler, Northwestern Regional Sales Manager 949.387.6684, jmahler@fawcette.com
- Chris Kourtoglou, Southern Regional Sales Manager 650.833.7155, ckourtoglou@fawcette.com

This listing is provided as a courtesy to our readers and advertisers. The publisher assumes no responsibility for errors or omissions.
Sales Department: 209 Hamilton Avenue • Palo Alto, CA, USA 94301-2500 • 650-833-7100

Continued from page 93.

```
    reply.getSOAPPart().getEnvelope(true);
envelope.addNamespaceDeclaration(
  "stock", "http://www.stocksrus.com/stock");

name = envelope.createName(
  "GetStockPrices", "stock");
child = body.getChildElements(name);
body = envelope.getBody();

if(!child.hasNext())
  throw new Exception();

getStockPrice = (SOAPElement)child.next();

name = envelope.createName(
  "stocks", "stock");
child = getStockPrice.
  getChildElements(name);

if(!child.hasNext())
  throw new Exception();

stocks = (SOAPElement)child.next();

name = envelope.createName(
  "stock", "stock");
child = stocks.getChildElements(name);

if(!child.hasNext())
  throw new Exception();

name = envelope.createName(
  "stocks", "stock");
stocks = body.addChildElement(name);

while(child.hasNext()) {
  String symbol, price;

  stock = (SOAPElement)child.next();
  name = envelope.createName(
    "symbol", "stock");
  stock = (SOAPElement)stock.
    getChildElements(name).next();
  symbol = stock.getValue();
  price = (String)quotes.get(symbol);

  if(price == null)
    price = "?";

  name = envelope.createName(
    "stock", "stock");
  stock = stocks.addChildElement(name);
  name = envelope.createName(
    "symbol", "stock");
  stock.addChildElement(
    name).addTextNode(symbol);
  name = envelope.createName(
    "price", "stock");
  stock.addChildElement(
    name).addTextNode(price);
 }

  reply.saveChanges();
 } catch(Exception e) {
  try {
    SOAPFault fault;
    reply = factory.createMessage();
    envelope =
      reply.getSOAPPart().getEnvelope(true);
    body = envelope.getBody();
    fault = body.addFault();
    // Would normally want to be more
    // informative than this.
    fault.setFaultCode("Server");
    fault.setFaultString(
      "Internal Server Error.");
    reply.saveChanges();
  } catch(SOAPException se) {
    return null;
  }
 }

  return reply;
 }
}
```

Part.setContent(Source). Rather surprisingly, there is no way to incorporate JAXB objects into JAXM messages directly. Given that programmers will want to bind the elements of JAXM messages to Java objects, you would think JAXM would support the insertion of JAXB objects into the SOAPMessage document hierarchy. The only apparent way to use JAXB with JAXM is to model the entire SOAPPart as a JAXB object (first creating the necessary DTD), marshal it, and bind the output to a StreamSource for use with setContent(Source). This is clearly less than ideal. Not interfacing well with JAXB also sacrifices the ability to validate a message before sending or upon receiving it. The example in Listing 3 makes a feeble attempt to ensure that the request is structured properly by checking at each step that the message has child nodes at expected locations. This wouldn't be necessary if it were possible to use JAXB more easily.

Where javax.xml.soap doesn't make things as easy as it should, javax.xml.messaging does better. The Endpoint class represents a communication endpoint, defined by a URI. The URI is bound to a transport protocol and address external to the program in a provider configuration file, provider.xml. The URLEndpoint class, used in QuoteClient in Listing 1, bypasses the provider and allows a direct connection to be specified with a URL. The JAXMServlet class makes it almost trivial to receive SOAP messages. It is designed to work with either the SyncListener or AsyncListener interfaces, each of which defines a method called onMessage(SOAPMessage), which JAXMServlet will invoke after receiving a message via the HTTP POST method. The SOAPMessage argument to onMessage will contain the message received, saving you from implementing your own doPost() method to convert the message. The SyncListener and AsyncListener interfaces appear to have been renamed ReqRespListener and OneWayListener in the final version of the JAXM specification, but the early access reference implementation uses the old names.

> **Where javax.xml.soap doesn't make things as easy as it should, javax.xml.messaging does better**

QuoteServlet extends JAXMServlet and implements the SyncListener interface and onMessage (see Listing 3). Implementing the SyncListener interface requires that onMessage return a SOAPMessage reply. The AsyncListener interface defines on-

Listing 4 SOAP Response

```
<Envelope xmlns="http://schemas.xmlsoap.org/
  soap/envelope/"xmlns:stock=
  "http://www.stocksrus.com/stock">
  <Header/>
  <Body>
    <stock:stocks>
      <stock:stock>
        <stock:symbol>TYR</stock:symbol>
        <stock:price>6</stock:price>
      </stock:stock>
      <stock:stock>
        <stock:symbol>ELL</stock:symbol>
        <stock:price>4</stock:price>
      </stock:stock>
      <stock:stock>
        <stock:symbol>HAL</stock:symbol>
        <stock:price>?</stock:price>
      </stock:stock>
    </stock:stocks>
  </Body>
</Envelope>
```

This is a formatted version of the SOAP reply generated by QuoteServlet.

Listing 5 Working with Tomcat

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//
  DTD Web Application 2.2//
  EN" "http://java.sun.com/j2ee/dtds/web-
  app_2_2.dtd">

<web-app>
<!-- Use this if writing a SOAP client servlet.
  <servlet>
    <servlet-name>initservlet</servlet-name>
    <servlet-class>
      com.sun.xml.messaging.
        client.local.InitServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
-->
  <servlet>
    <servlet-name>quoteexample</servlet-name>
    <servlet-class>
      example.QuoteServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
<!--
  <servlet-mapping>
    <servlet-name>initservlet</servlet-name>
    <url-pattern>/client</url-pattern>
  </servlet-mapping>
-->
  <servlet-mapping>
    <servlet-name>quoteexample</servlet-name>
    <url-pattern>/getquote</url-pattern>
  </servlet-mapping>
</web-app>
```

A JAXM servlet that provides only a service does not require any special setup, but a servlet that acts as a client requires special initialization in its web.xml file to work with Tomcat in addition to client.xml and provider.xml files. The web.xml file shown here has client initialization commented out.

Message with a void return type and requires a reply, if any, to be sent independently. QuoteServlet parses queries like the one in Listing 2 and produces a response like the one in Listing 4, containing the prices of the requested stocks. JAXMServlet does all the communication work in its implementation of doPost(), so QuoteServlet only disassembles a query and assembles a reply using the javax.xml.soap package. QuoteClient communicates with QuoteServlet using javax.xml.soap.SOAPConnection. SOAPConnection's call method requires a SOAPMessage and an Endpoint to send a message. It might have been better for JAXM to provide a generic Connection class or interface in the messaging package and to define a generic Message interface, implemented by SOAPMessage, to allow messaging standards other than SOAP to be handled through the same interfaces. Right now, JAXM is very much a SOAP-only API.

Listing 5 contains a sample web.xml file for deploying QuoteServlet with Tomcat. When using a provider, you need to load a provider servlet using the commented-out deployment markup. The provider will receive asynchronous replies for client requests and also generate appropriate MessageFactory instances for specific profiles. The profile used by a client connection is specified in a client.xml configuration file. Some of the examples provided with the JAXM reference implementation demonstrate how to use an ebXML profile. JAXM can also be used from JavaServer Pages (JSP). The JAXM tag library makes it easier to work with JAXM in JSP than in servlets. SOAP message bodies can be inlined, removing the hassle of using the javax.xml.soap classes. However, it is questionable from a software maintenance perspective whether writing Web services in JSP pages is a good thing to do.

Even though JAXM brings some structure to writing Web services in Java, it does not integrate well with JAXB. As long as the Java APIs for XML remain relatively disjointed, you will have to write extra code to do the things you want. Nonetheless, just as EJB didn't really get rolling until version 2.0, Java Web services may need to wait for a second revision of JAXM and other specifications before they can take off. In the meantime, JAXM helps solidify the definition of Web services in the Java 2 Platform. JP

Daniel F. Savarese is an independent software developer and technology advisor. He has been the founder of ORO Inc., a senior scientist at Caltech's Center for Advanced Computing Research, and vice president of software development at WebOS. Daniel is the original author of the Jakarta-ORO text processing packages and the NetComponents network protocol library. He is also a coauthor of *How to Build a Beowulf* and is working on several computer science book projects. He can be reached at **dfs@savarese.org**.

Use these Locator+ codes at www.javapro.com to go directly to these related resources.

JP0201 Download all of the code for this issue of *Java Pro*.

JP0201PS Download the code for this article separately.

JP0201PS_T Read this article online.

Java & XML, 2nd Edition

by Brett McLaughlin

AUTHOR: Brett McLaughlin
PUBLISHER: O'Reilly & Associates Inc.
ISBN: 0-596-00197-5
PAGES: 528 pages
PRICE: $44.95

Few combinations are as powerful as Java and XML and, although a few books have been written about this topic, it's hard not to recommend this one as a great investment. *Java & XML* covers the basics of Java's XML APIs, including DOM and SAX, across multiple versions. The JDOM API is well explored, as are XML-based solutions like XML-RPC, SOAP, UDDI, WSDL, and more. If you're working with Java and XML, this book is a great source of information, examples, explanations, and practical advice.

After a brief introduction and orientation in the first two chapters, the author jumps right to the first API and provides a pair of chapters for SAX, DOM, and JDOM respectively. It's interesting that the author chose to cover SAX before the DOM API, but perfectly logical when you consider the approach taken in the coverage for each. SAX is arguably simpler to learn and put to work. The DOM coverage is good, but I was especially happy to see the JDOM API given as much attention as both SAX and DOM. For each of these APIs, the book starts with a basic introductory chapter, followed by an advanced chapter that explores more powerful, perhaps less frequently used, capabilities.

Chapter 9 presents the JAXP approach introduced by Sun, complete with coverage of its various permutations and what you can expect in each. In fact, this book is especially good at providing information about where you might get caught by unexpected behaviors or variants, and what you need to pay special attention to in order to make sure everything is functional. The rest of the book takes a turn toward higher-level solutions.

After a look at Web publishing frameworks, like Apache's Cocoon, we move on to XML remote procedure calls (XML-RPC) and Web services using SOAP. One chapter takes a close look at the SOAP protocol, and a subsequent chapter explores what it takes to expose Web services using UDDI and WSDL. Chapter 14 provides an exploratory look at content syndication, followed by a chapter that covers current data binding solutions. The book closes with a look ahead to XML efforts.

This is a pragmatic book written with a simple philosophy of facilitating the programmer experience with Java and XML. The author makes a point of exploring only final releases, leaving unfinished APIs for future editions. However, the publisher seems to have gotten a little carried away with the hype on the back cover, promising a look at XPath support, which is little more than a path generation example, rather than the implied XPath search facilities now available. With that minor exception, the book lives up to its promise and makes for a good addition to any serious Java developer's bookshelf.

–Claude Duguay

Java and XSLT

by Eric M. Burke

AUTHOR: Eric M. Burke
PUBLISHER: O'Reilly & Associates Inc.
ISBN: 0-596-00143-6
PAGES: 528 pages
PRICE: $39.95

Where Java and XML have become an unbeatable pair, more and more Web-based application developers are discovering the benefits of XSLT in keeping data formats and presentation layers as uncoupled as possible. XSLT is designed to easily transform XML documents and messages from one format to another. If you can produce source materials in XML, they can be delivered to virtually any platform with the help of XML style-sheet transformations.

In *Java and XSLT*, the first few chapters lay down a solid foundation, with a brief introduction and a well-explored look at XSLT transformations. The overview includes a quick look at the style-sheet syntax, XPath basics, looping, sorting, conditional processing, variables, formatting, and schema style sheets. Of particular interest in this part of the book are techniques for dynamically setting XSLT attributes programmatically—one of those little tricks that can make a huge difference in the way you approach XSL solutions with Java.

Much of this book centers on Web applications, so it includes a fair amount of discussion about everything from the Model-View-Controller (MVC) architecture and the use of servlets and JavaServer Pages (JSP) to using or generating XML source documents for publishing. The book touches on using Enterprise JavaBeans (EJB) with XSLT, as well as the use of JAXP to gain access to solution-independent parsers and XSL template handlers.

By the time you hit Chapter 6, you're getting information about WAR files and Web application development with the servlet API. Because templates can be compiled, the chapter includes a good example of caching servlets for faster processing. Chapter 7 includes a good example application in the form of a discussion forum. This is followed by a chapter that looks at slightly more advanced techniques using servlet sessions and browser identification, among other useful tips. The book wraps up with a look at deployment, concluding with a chapter on wireless applications.

It's hard not to like this book. The author takes a bold stand on using XSLT as a vehicle for Web application delivery and provides a complete example that shows how to use the technology effectively. There are plenty of other ways to approach Web application development today, and various frameworks from which to choose, so it takes strong conviction to make this solid of a case for XSLT. The material is well explored and includes more than a few information nuggets for the well-informed programmer. And at $5 less than O'Reilly's companion book, *Java & XML*, but with the same page count, the price is attractive too.

–Claude Duguay

sitraka

JProbe™

Because performance is mission critical.

JProbe ServerSide Suite
The ultimate solution for building performance and reliability into your server-side applications.

JProbe Profiler (with integrated Memory Debugger)
Pinpoint and eliminate performance bottlenecks and inefficient memory usage.

JProbe Threadalyzer
Track down threading hazards including deadlocks, stalls and race conditions.

JProbe Coverage
Don't let untested code put your J2EE apps at risk. Find and quantify untested code to ensure thorough and accurate test runs.

JProbe Training
Learn best practices in developing for maximum Java performance from instructors who understand your development challenges.

Speed. Reliability. Rock-solid code.

J2EE applications need it all. That's where Sitraka JProbe comes in. JProbe integrates with all major application servers for fast, accurate analysis of your servlet, JSP and EJB code.

JProbe supports Microsoft Windows, Sun Solaris, RedHat Linux, HP-UX and IBM AIX.

Visit our integration portal for J2EE and download a free eval now

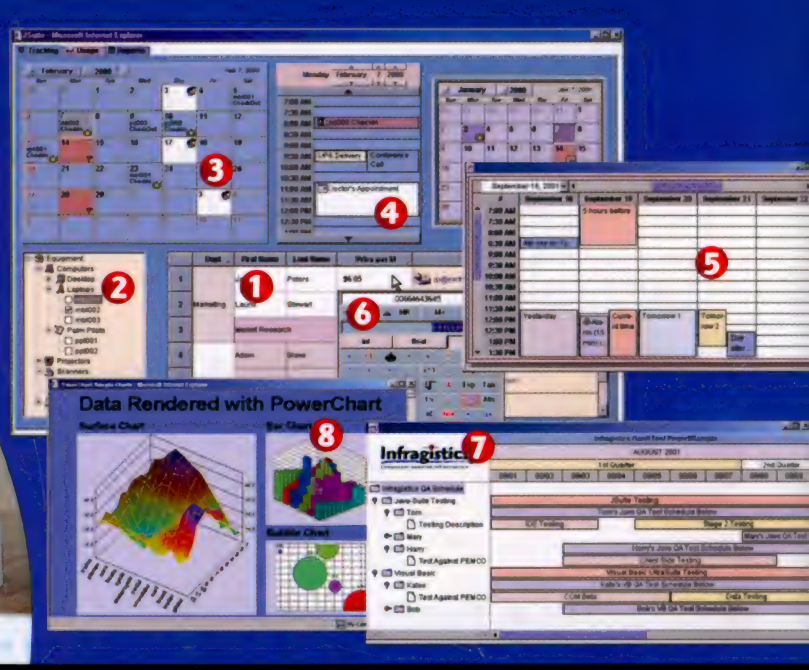
www.sitraka.com/jprobe/jp

1 888-361-3264 | Europe +31(0)20 510 67 00

sitraka

the Java™ advantage

© Copyright 2001 by Sitraka Inc. All rights reserved. Sitraka, the Sitraka logo and JProbe are trademarks of Sitraka Inc. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in US and other countries. All other products are trademarks or registered trademarks of their respective companies.



JSuite™ 6.0–Total Java Solution

Infragistics - formed by the merger of Sheridan and ProtoView

Packaged To Please.

JSuite provides the most comprehensive set of Java-based components available today. Now we've combined our Java suites to deliver it all: AWT, JavaBeans™ and Swing. Create powerful, eye-pleasing applications or browser-based applets with the rich look and feel your users expect, and our new server features add robust back-end functionality to your applications.

- **NEW** Gantt–Display hierarchical data in the familiar style of a Gantt chart (tree nodes associated with bar chart data display).
- **NEW** WeekView–Display and edit appointments for multiple days.
- **NEW** Password–Authenticate user ID and password against a directory server or database, with encryption.
- **NEW** Access Control List– Regulate permissions and levels of access based on roles assigned to your application/system user's logon.
- **NEW** Connection Pooling–Maintain a specified pool of connections, eliminating the need for time-dependent 'retry connection' code.
- Data Models™ provides automatic data synchronization between our chart and grid with our exclusive AutoSynch™ technology.
- Client-side charting delivers stunning 2D/3D data visualization in all major chart types,with instantaneous databinding and dynamic update capabilities.

Enterprise Editions include the product, annual subscription service, GUARANTEED priority response for support, and many other special advantages. **Corporate programs** for large development teams

1. Grid–Extends JTable
2. Tree–Extends JTree
3. Calendar–Includes DayView & Date edit
4. DayView
5. **NEW** WeekView
6. Calculator
7. **NEW** Gantt
8. Charting–Client-side

JSuite 6.0 includes:

- Applet Wrappers
- Calendaring
- Charting
- Advanced UI Components
- Data Explorer
- Editing Components
- Grid
- Tab
- Tree
- Utility Components
- And More!!

For detailed information, visit
www.infragistics.com
Download Free Trial Version!
Order Online! or call 800-231-8588

Infragistics™
Component powered infrastructure

Copyright 2001 Infragistics, Inc. All rights reserved. Infragistics, the Infragistics logo, JSuite, Data Models, ActiveImage & AutoSynch, Applet Wrappers & PowerChart are trademarks of Infragistics, Inc. JavaBeans is a trademark of Sun Microsystems. All other trademarks or registered trademarks are the respective property of their owners.

Java • ActiveX • .NET 800-231-8588 • infragistics.com

One Year ONLY $19.95!

74% off the cover price of JAVAPro

Name:

Company Name:

Address:

City State: Zip:

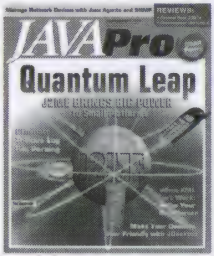
Country:

E-mail (required):

One year of *Java Pro* is 12 issues plus one special issue. Savings based on the annual cover price of $77.35. Please allow four-to-six weeks for delivery of the first issue. International subscriptions must be prepaid in U.S. funds plus postage: Canada and Mexico add $18 per year for surface delivery; all other countries add $44 per year for airmail delivery. New orders only.

☑ YES!

Start my subscription to *Java Pro* and bill me for just $19.95



JAVAPro

421D

Subscribe Online: http://www.javapro.com

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 317 PALO ALTO, CA

POSTAGE WILL BE PAID BY ADDRESSEE

JAVAPro™

PO Box 59277
Boulder, CO 80323-9277

Choosing an SCM solution?

Get your free SCM evaluation guide today.

Just complete and return this reply card. Or visit www.rational.com/offer/scmguide15.

☐ YES! Send me the free "Beyond Version Control" guide to choosing an SCM solution.

☐ No thanks. We're into chaos.

NAME _____

JOB TITLE _____

COMPANY _____

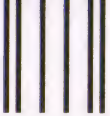
ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____

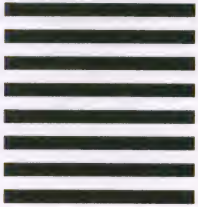
PHONE _____ E-MAIL ADDRESS _____

JP-AC-CCAD-D1

Rational
the software development company



NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 9556 LEXINGTON MA

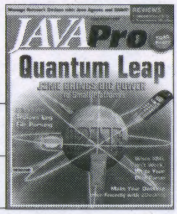
POSTAGE WILL BE PAID BY ADDRESSEE

RATIONAL SOFTWARE
20 MAGUIRE ROAD
LEXINGTON MA 02421-9566

One Year ONLY $19.95!

JAVAPro™

74% off one year of Java Pro!



Name:

Company Name:

Address:

City State: Zip:

Country:

E-mail (required):

☑ YES!

Start my subscription to *Java Pro* and bill me for just $19.95

One year of *Java Pro* is 12 issues plus one special issue. Savings based on the annual cover price of $77.35. Please allow four-to-six weeks for delivery of the first issue. International subscriptions must be prepaid in U.S. funds plus postage: Canada and Mexico add $18 per year for surface delivery; all other countries add $44 per year for airmail delivery. New orders only.

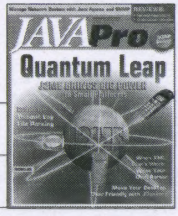
421N

Subscribe Online: http://www.javapro.com

One Year ONLY $19.95!

JAVAPro™

74% off one year of Java Pro!



Name:

Company Name:

Address:

City State: Zip:

Country:

E-mail (required):

☑ YES!

Start my subscription to *Java Pro* and bill me for just $19.95

One year of *Java Pro* is 12 issues plus one special issue. Savings based on the annual cover price of $77.35. Please allow four-to-six weeks for delivery of the first issue. International subscriptions must be prepaid in U.S. funds plus postage: Canada and Mexico add $18 per year for surface delivery; all other countries add $44 per year for airmail delivery. New orders only.

421N

Subscribe Online: http://www.javapro.com

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 317 PALO ALTO, CA

POSTAGE WILL BE PAID BY ADDRESSEE

JAVAPro™

PO Box 59277
Boulder, CO 80323-9277

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 317 PALO ALTO, CA

POSTAGE WILL BE PAID BY ADDRESSEE

JAVAPro™

PO Box 59277
Boulder, CO 80323-9277

FREE SUBSCRIPTION TO XML MAGAZINE!

BUILDING THE NEXT GENERATION INTERNET

❑ **YES!** I would like to receive *XML Magazine* free of charge!

❑ **No** I would not like to receive *XML Magazine* free of charge!

Signature *(required)*

Date *(required)*

Name

Title

Company

Address

City State Zip Country

Phone Fax

E-mail Address *(required)*

❑ Please do not use my email to contact me about my subscription status and related Fawcette products F1JN

❑ Please do not send me third party email messages about products and services from *XML Magazine* partners

Subscriptions are free to qualified IS Professionals in the U.S. Paid subscriptions are available for non-U.S. readers. International subscriptions are to be prepaid in U.S. funds; Canada/Mexico $64.00, all other $84.00.

For faster service renew online at www.xml-mag.com/B1CN

or fax to: 925/372-8582

Please answer all questions, sign and date. Incomplete forms cannot be processed.

1. PRIMARY BUSINESS ACTIVITY AT YOUR LOCATION (SELECT ONLY ONE):

101 ❑ Finance, banking, accounting, insurance
102 ❑ Manufacturing other than computer hardware and software and software
103 ❑ Manufacturing computers and computer software
104 ❑ Non-computer retailer, distributor, wholesaler
105 ❑ Applications software development
106 ❑ Aerospace
107 ❑ Government, military
108 ❑ Agriculture, mining, construction, oil & gas
109 ❑ Transportation, utilities
110 ❑ Media, marketing, advertising
111 ❑ Real estate
112 ❑ Research and development
113 ❑ Engineering
114 ❑ VAR/Systems Integrators
115 ❑ Communications services
116 ❑ Computer-related retailer, distributor, wholesaler
117 ❑ Systems software development
118 ❑ Education
119 ❑ Health, medical, legal
120 ❑ Consulting (please specify)

199 ❑ Other (please specify)

2. WHAT IS YOUR CURRENT JOB TITLE? (SELECT ONLY ONE FROM THE FOLLOWING SIX (6) CATEGORIES):

Application development manager/ architect:

201 ❑ Applications development manager
202 ❑ Software development director
203 ❑ Software architect
204 ❑ Software project leader
205 ❑ Project manager
206 ❑ Advanced technology evaluator
207 ❑ Software team leader
208 ❑ Software engineering supervisor
209 ❑ Systems analyst/designer
210 ❑ Database designer
211 ❑ Application manager/administrator
295 ❑ Other (please specify)

IS manager:

230 ❑ CIO, Technical VP
231 ❑ IS/MIS manager
296 ❑ Other (please specify) 4/01

Over Please ☞

Application developer:

240 ❑ Applications developer/ programmer
297 ❑ Other (please specify)

Computer operations staff:

250 ❑ System Administrator
251 ❑ LAN Administrator
252 ❑ Help Desk
298 ❑ Other (please specify)

Corporate manager:

260 ❑ CEO, president, partner, chairman, owner
261 ❑ VP, controller, director of marketing/sales, general manager, division/department head

Other (categories not listed above):

299 ❑ Other (please specify)

3. HOW ARE YOU CURRENTLY USING XML?

301 ❑ Currently using
302 ❑ Planning to use XML in the next 12 months
303 ❑ Evaluating XML for applicability
304 ❑ Not planning on using XML

4. ARE YOU DEVELOPING APPLICATIONS THAT WILL INVOLVE ONE OR MORE SERVERS OR HOSTS OR DO YOU PLAN TO IN THE NEXT 12 MONTHS?

401 ❑ Yes
402 ❑ No

5. SERVER/HOST OPERATING SYSTEMS IN USE AT THIS ENTIRE LOCATION (PLEASE CHECK ALL THAT APPLY):

501 ❑ Windows NT/2000 Server
502 ❑ Unix (all flavors)
503 ❑ Netware/Intranetware
504 ❑ OS/400
505 ❑ VMS
506 ❑ MVS/VM
507 ❑ OS/2
599 ❑ Other (please specify)

6. WHAT IS THE ANNUAL DOLLAR VALUE OF COMPUTER PRODUCTS AND SERVICES FOR WHICH YOU ARE PERSONALLY INVOLVED IN THE PURCHASING DECISIONS FOR YOUR ORGANIZATION'S OR LARGEST CLIENT'S INTERNAL USE?

601 ❑ More than $10 million
602 ❑ $5 million - $9,999,999
603 ❑ $2 million - $4,999,999
604 ❑ $1 million - $1,999,999
605 ❑ $500,000 - $999,999
606 ❑ $100,000 - $499,999
607 ❑ $50,000 - $99,999
608 ❑ $10,000 - $49,999
609 ❑ Less than $10,000

7. WHAT IS THE NUMBER OF EMPLOYEES OF THE LARGEST BUSINESS ENTITY BY WHOM YOU ARE EMPLOYED?

701 ❑ Under 100
702 ❑ 100 - 999
703 ❑ 1,000 - 4,999
704 ❑ 5,000 - 9,999
705 ❑ 10,000 - 49,999
706 ❑ More than 49,999

Free subscription

FOLD AND TAPE, DO NOT STAPLE

PO Box 2507
Martinez, CA 94553-9917

XML MAGAZINE

POSTAGE WILL BE PAID BY ADDRESSEE

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 317 PALO ALTO, CA

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES